TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ELLI HEIKKINEN

TESTING METRICS IN A SOFTWARE DEVELOPMENT COMPANY

Master of Science thesis

Examiner: Prof. Hannu-Matti Järvinen
Examiner and topic approved by the
Faculty Council of the Faculty of
Computing and Electrical Engineering
on 14th January 2015

ABSTRACT

ELLI HEIKKINEN: Testing metrics in a software development company
Tampere University of Technology
Master of Science thesis, 66 pages, 3 Appendix pages
May 2015
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Hannu-Matti Järvinen
Keywords: software testing, software quality, testing process, test management, metrics, measurement

Software testing is an important part of the software development process. Its purpose is to provide information about the quality of software. This information is often used in decision making and in improving software quality. When tested systems grow in size and complexity, testing gets difficult to manage in terms of scheduling, prioritising, monitoring and reporting, for example. For test management to be effective, great effort is needed. However, without proper metrics, test management is difficult. The primary goal of this thesis was to develop a basic set of testing metrics to aid in test management of a software company called M-Files that develops an enterprise content management (ECM) system carrying the same name as the company. Additionally, the secondary goal was to identify ideas for metrics that help improving testing and product quality.

Literature was reviewed for information on testing metrics, instructions on how testing metrics should be developed and what kind of metrics are popular in software development field. A process standard IEEE Std 1061-1998 (R2009) Software Quality Metrics Methodology standard for developing software quality metrics was partially applied in this thesis. Also, the current testing and test management practices in the case organisation were studied and their problems in test reporting were further analysed. Semi-structured interviews and periodic reviews were used to identify and refine metrics requirements in the organisation. From the requirements, a set of metrics was defined, prioritised, and implemented with M-Files Reporting tool. M-Files Reporting tool was chosen because it allows automatic data extraction, existing tools for data analysis and visualisation, and a way to deliver reports easily through the organisation.

As a result, a set of seven basic metrics and their representations in reports were implemented. The developed reports answer the organisations urgent needs for test management and reporting support. Also, several metrics requirements and improvement points for future development of processes and tools were identified. This thesis is a first step of a more fundamental metrics and process development task. Along with the developed metrics, the organisation gained a lot of useful knowledge about metrics and their development process for future utilisation.

TIIVISTELMÄ

ELLI HEIKKINEN: Testauksen metriikat ohjelmistoyrityksessä
Tampereen teknillinen yliopisto
Diplomityö, 66 sivua, 3 liitesivua
Toukokuu 2015
Tietotekniikan koulutusohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: Professori Hannu-Matti Järvinen
Avainsanat: ohjelmistojen testaus, ohjelmiston laatu, testausprosessi, testauksen hallinta, metriikat, mittarit, mittaaminen

Testaus on tärkeä ja välttämätön osa ohjelmiston kehitystyötä. Sen tarkoitus on antaa tietoa testattavan järjestelmän laadusta. Tätä tietoa käytetään usein päätöksenteossa ja testattavan järjestelmän laadun parantamisessa. Kehitettävien järjestelmien kasvaessa ja monimutkaistuessa, testauksen hallinta vaikeutuu muun muassa aikataulutuksen, priorisoinnin, seurannan ja raportoinnin suhteen. Jotta testaus olisi tehokasta ja tuottaisi tulosta, vaaditaan testauksen hallinnalta suuria ponnisteluja. Kuitenkin, ilman kunnollisia mittareita, testauksen hallinta on hankalaa. Tämän työn ensisijaisena tavoitteena on kehittää testauksen hallinnassa auttavia mittareita M-Files nimiselle ohjelmistoyritykselle, joka kehittää samannimistä yritystiedon hallintajärjestelmää. Lisäksi toissijaisena tavoitteena on löytää yritykselle ehdotuksia tulevaisuudessa kehitettävistä mittareista, joiden avulla testausta ja lopulta tuotteen laatua voitaisiin parantaa.

Kirjallisuuskatsauksessa tutkittiin testauksen mittareista yleensä, ohjeita siihen kuinka testauksen mittareita tulisi kehittää ja mitkä mittarit ovat suosittuja ohjelmistokehityksen alalla. Työssä metriikoita kehitettiin soveltuvin osin IEEE 1061-1998 (R2009) prosessistandardin mukaisesti. Työssä tutkittiin myös yrityksen nykyisiä testaus- ja testauksen hallintakäytäntöjä ja analysoitiin testauksen raportoinnin ongelmia. Yrityksen tarvittavien mittareiden tunnistamiseen käytettiin teemahaastatteluja ja niiden yksityiskohtia tarkennettiin katselmuksilla. Haastatteluissa löydetyistä vaatimuksista kehitettiin priorisoitu lista kehitettävistä mittareista, jotka toteutettiin M-Files-ohjelmiston raportointityökalulla. Raportointityökalu valittiin, koska sen avulla datan kerääminen voitiin automatisoida, datan analysointiin ja visualisointiin oli valmiit työkalut ja koska raporttien jakaminen koko organisaatiolle oli helppoa.

Tässä työssä kehitettiin yhteensä seitsemän mittaria valmiiksi visualisoiduiksi raporteiksi. Kehitetyt raportit vastaavat yrityksen kiireellisimpiin tarpeisiin testauksen hallinnassa ja tulosten raportoinnissa. Lisäksi, useita muita mittareita ja työkalujen ja prosessien parannusideoita kehitettiin tulevaisuutta varten. Tämä työ oli ensimmäinen osa kokonaisvaltaisempaa metriikoiden ja prosessien kehitystyötä M-Filesillä. Kehitettyjen metriikoiden lisäksi, tämän työn myötä yrityksen ymmärrys testauksen metriikoista ja niiden kehittämisprosessista kasvoi, mistä on varmasti hyötyä metriikoiden kehittämiseen myös tulevaisuudessa.

PREFACE

Firstly, I would like to thank Minna Vallius and Veikko Juusola for giving me this subject and the opportunity to do this thesis at M-Files. Thanks for Minna for also reviewing and supporting my work. Doing this thesis in a practical environment gave this work an actual value.

I would like to thank Professor Hannu-Matti Järvinen for his guidance with my thesis. Special thanks to Matti Vuori for providing me with numerous valuable suggestions and comments. It would have been a lot more difficult without him. Warm thanks to Antti for supporting me and putting up with me when I spent all my time working with this thesis. Finally, I would like to thank everyone in the R&D department for their encouragement.

Tampere, 16.04.2015

Elli Heikkinen

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

DoD	Definition of Done
DRE	Defect Removal Efficiency
EMC	Enterprise Content Management
FP	Function Point
GQM	Goal/Question/Metric methodology
IE	Internet Explorer web browser
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Standards Organization
ISTQB	International Software Testing Qualifications Board
LOC	Lines of Code
QMS	Quality Management System
R&D	Research and Development
ROI	Return On Investment
TMMi	Test Maturity Model Integrated (software testing)
UI	User Interface
URL	Uniform Resource Locator

1. INTRODUCTION

Software testing is an essential part of software engineering. The objective of testing is to give insight into the quality of the software (ISO/IEEE/IEC 2013). That information is usually utilised in decision making and in improving the product quality. However, as the tested systems grow in size and complexity, the same happens to testing and the pool of data it provides. For testing to be efficient and productive, extensive management effort and good testing practices are required. Metrics are a powerful and even mandatory tool for both effective test management and extracting the test results for decision making. Metrics provide objective and exact information which is mandatory for having control. There are two phrases that are often quoted as justifications for using metrics as tools of control. The first one is: "You can't control what you can't measure" by Tom DeMarco and the other is: "when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind..." by William Thomson, also known as Lord Kelvin, as quoted in (Craig & Jaskiel 2002, pp. 371-372). The same concepts are mentioned in various formats in other publications as well (Walia 2012, Crispin & Gregory 2009, Lazic & Mastorakis 2008).

Without metrics, it is difficult to plan testing, monitor testing progress, and report the test results. These difficulties in test traceability and testing reporting have been noted in a Finnish software company M-Files which has experienced rapid growth in the past few years. As their product and the number of developers have grown, test management and reporting have turned out too laborious or even impossible when using the current tools and methods. These problems surface especially during system testing. As a result, a new test management tool that offers better test asset management and test execution capabilities was recently taken into operation in the organisation. Still, even with the new tool, test planning, scheduling, and monitoring requires manual data collection from several sources. Due to issues with previous testing tools, the number of automated testing metrics in the organisation is practically zero. Because the new tool made data collection easier to automate, the topic of developing testing metrics arose.

It is often difficult to define which metrics to use (Vuori 2014). Ultimately, testing metrics can help guiding the test efforts to reach the previously set quality criteria. However,

it has been acknowledged in the case organisation that before any advanced metrics can be created, it is necessary to have the basics of test management in adequate condition. Only after that, metrics can be created for improving testing. At the moment, the organisation is still developing their test management practices and tool support with scarce resources, while even the most basic metrics still require considerable manual effort to collect. Therefore, the objective at this phase is in improving M-Files' test management tools so that less effort is needed in test planning and monitoring and more effort can be directed to reaching the product quality requirements. The primary goal of this study is therefore to implement automated collection and representation of the most basic testing data available in M-Files' test management tool. The secondary goal is to identify metrics and ways to develop the testing tools and practices in the organisation so that the metrics could be used to guide the testing efforts for improved software quality.

The first chapters of this thesis feature the background of metrics and reporting in the field of software testing. Chapter 2 includes an introduction to the essential parts of software testing and test management and a review to the goals and limitations of testing metrics. In Chapter 3, a detailed overview of a process standard for developing metrics in an organisation is given. Additionally, a few popular metrics in the software development field are introduced. Chapter 4 focuses on the case organisation, giving a short introduction to the organisation and its product development process. After that, the tasks in the testing process and test management of the organisation are explained, including test case and defect management, test planning, and test monitoring. Finally, the organisation's current testing metrics and the problems in reporting are discussed in detail. Chapter 5 describes the whole process of identifying and developing metrics in the organisation, including a review and evaluation of the resulting metrics in the end. The used methods follow the recommendations of the metrics development process introduced in Chapter 3. The metrics requirements are identified with semi-structured interviews and further analysed and prioritised for implementation. The chapter includes a description of M-Files (software) Reporting feature which was used in implementing the metrics. In Chapter 6, the results as well as the quality and limitations of this study are evaluated. Also, some implications for future development are discussed. Finally, Chapter 7 concludes the thesis with a brief summary of the conducted research.

2. METRICS AND REPORTING IN SOFTWARE TESTING

Metrics are an important part of software development, testing, and test management. For effective test management, information about the testing progress, remaining effort to be done, and the overall test process is required. In IEEE Standard for a Software Quality Metrics Methodology (IEEE 1998), the purpose of software quality metrics is defined as "to make assessments throughout the software life cycle as to whether the software quality requirements are being met". In other words, testing metrics should provide objective and quantified information about the quality of the product. Agarwal et al. (2010, p. 149) state that "metrics-based management is also a key component in the software engineering strategy for achieving its objectives". Without proper metrics, test management and decision making rely on individuals' experience and subjective opinions, which can be considered going against the fundamentals of developing quality software.

However, metrics are not without limitations and challenges. Collecting and analysing data might be very time consuming and costly. Then, the data should be presented in a way that the audience can understand it and not misinterpret it. Also, even if the information was conveyed successfully, the collected data might not represent the object being measured. Finally, wrong things might be measured. Despite these limitations, measuring is still better than not measuring at all, just as the two quotes above suggest. It is possible to create effective metrics as long as their limitations are kept closely in mind when implementing metrics in an organisation. (Craig & Jaskiel 2002, Crispin & Gregory 2009)

Before going further into testing metrics, a brief introduction to software testing and test management that necessitate testing metrics altogether is given. As the following section shed light on the complexity of test context and its management, the importance of testing metrics becomes clearer. After that, the purpose and goals as well as the challenges of test metrics are reviewed.

2.1 Software testing

There are many definitions for software testing. It can be defined as a process of software engineering in which the software is executed with the intent of finding errors. Alternatively, it can be defined as "the process of analysing a software item to detect the differences between existing and required conditions (i.e., bugs) and to evaluate the features of software items" (Agarwal et al. 2010, p. 162). Most of the definitions describe testing as a process rather than an activity. This means that instead of a single task, testing is a series of interrelated or interacting activities that lead to a particular result (ISO/IEEE/IEC 2013). Also, the definitions often emphasize that the goal of testing is to "provide information about the quality of the test item" (ISO/IEEE/IEC 2013) in terms of both functional and non-functional requirements. Software quality on the other hand can be defined as "the degree to which software possesses a desired combination of attributes" (IEEE 1998). Software quality attributes are visualised in Figure 2.1.

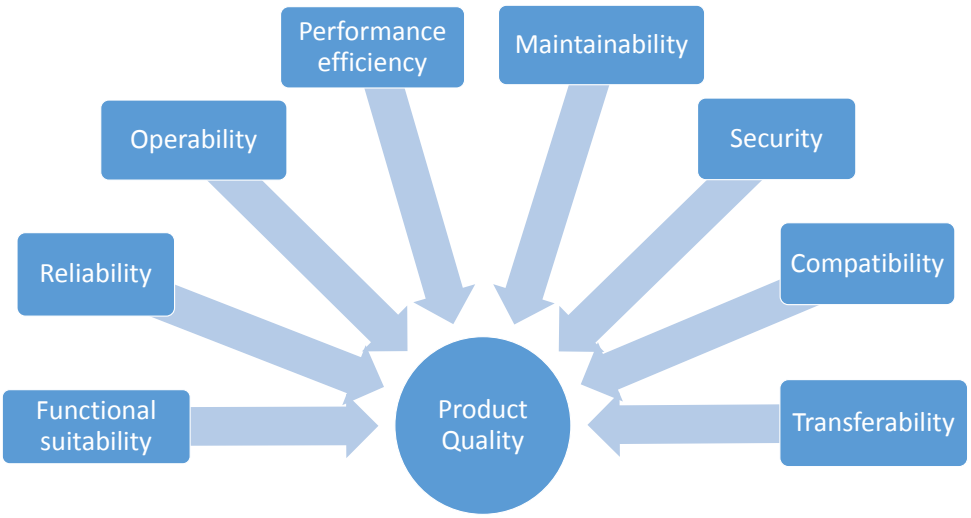


Figure 2.1 *Software product quality is a combination of eight attributes according to the ISO/IEC 25010 standard. (ISO 2011)*

It is a generally accepted fact that a human being can make an error, which then results in a *defect* in the product. A defect may be behaviour in an unintended way or produce an incorrect or unexpected result (Müller & Friedenberg 2011). According to ISO/IEEE/IEC (2013), the purpose of testing is to (1) provide information about the quality of the software, its risk of failure during use, and (2) find defects before it is released to the market. However, one of the benefits of testing is also increasing understanding of the system, especially when developing a very complex system. As a summary, the purpose of testing can be phrased with five objectives:

1. Finding defects to help improve the level of quality.
2. Reducing the risk of failures occurring during operation and gain confidence about the level of quality.
3. Improve management decisions by providing information for decision making.
4. Prevent defects by identifying the processes in the organization that need improvements.
5. Gain insight into the system behaviour.

It is easy to see that testing has become an integrated part of software engineering (Parveen et al. 2007).

At this point, it is necessary to clarify the two terms associated with testing: *verification* and *validation*. They are often and easily confused with each other but they do not mean the same thing. Verification is defined as confirming that the product meets the its functional and non-functional requirements and answers to the question "are we building the product right". Validation, on the other hand, is a more general process of evaluating whether the product meets the needs of the customer or other stakeholders and answers the question: "are we building the right product?". Validation is ensuring that the software is doing what the customer really needs. (Agarwal et al. 2010, pp. 85-86) It should be noted that not all testing is functional testing but there are usually has several other quality characteristics that are important to the stakeholders. Testing prior release may include testing other, non-functional, requirements such as performance, reliability, usability, and security. (Müller & Friedenberg 2011, Black et al. 2012)

There are different levels or sub-processes of testing that can be done. The usually mentioned test levels are unit testing, integration testing and system testing. Test levels are also typically called test phases, test stages, or test tasks (ISO/IEEE/IEC 2013). The levels differ in their objectives, the test object being tested, and the typical defects that the testing should find. Unit testing is the testing individual components independently from the other components, such as modules or classes. Stubs and drivers may be used for simulating the other components. Unit level testing is done before integrating with other system components because it is easier to discover errors in a small unit without any complicated interactions. Unit testing is a good practice to ensure that the unit works as required before combining all units. With small enough units, one may also attempt to test in an exhaustive fashion to reach confidence of the unit quality. (Agarwal et al. 2010, p. 165)

Integration testing is the second level of testing where individual modules are combined into subsystems in order to find errors in their interfacing. The term integration testing may refer to at least two different type of testing with different objectives: testing the in-

teractions between individual components or testing the system's interaction between different systems or hardware. (Müller & Friedenberg 2011) There are various approaches to integration testing including regression testing and smoke testing among others. (Agarwal et al. 2010, p. 168).

The third and final main level of testing is system testing which occurs when all subsystems are integrated into a complete system. This level of testing aims at finding errors that result from more complicated interactions between subsystems and other system components. System testing concerns more with the validation testing of the system than the other two levels. (Agarwal et al. 2010)

There are several techniques to testing. One especially worth mentioning is exploratory testing that is defined as a "simultaneous learning, test design and test execution" (Bach 2003). Exploratory testing is especially suitable for agile software development. This is because test design and execution are performed at the same time, which leaves more time for test planning. It also allows testers to use their intelligent mind to concentrate on the areas they find critical and less so on other areas. This responsibility may make testers observe the tested systems more carefully than when just executing prescribed test cases. Its downsides are that the results depend on the skills of the person conducting testing. Exploratory testing requires various skills of testers: critical thinking, observation skills, and the ability to design tests that systematically explore the product while also producing creative test scenarios on the go. (Bach 2003) Exploratory testing is sometimes mistaken as ad hoc testing, which is incorrect. While exploratory testing is guided by the testers' observations made of the tested system during testing, ad hoc testing is completely unplanned and unstructured testing. (Craig & Jaskiel 2002, pp. 177-178)

The context of testing is formed of various factors out of which testing levels and techniques are just a part of. Some of these are visualised in Figure 2.2. External factors such as important customers or rules, regulations, standards, and laws define the environment the organisations have to operate. Based on their operating environment, organisations have to develop their test strategy to meet the requirements. A test strategy may consist of factors such as test data (test cases and test environments), testing techniques and practices, test automation strategy, and defect management. Following their strategy, the organisations create test plans to test their system. The complex nature of the test context requires test management which should create control in the complex environment. (ISO/IEEE/IEC 2013)

As the software systems being developed get more complex and people are expected to produce even larger, robuster, and more reliable software with less time and less resources, effective management effort is required to manage the development process, including

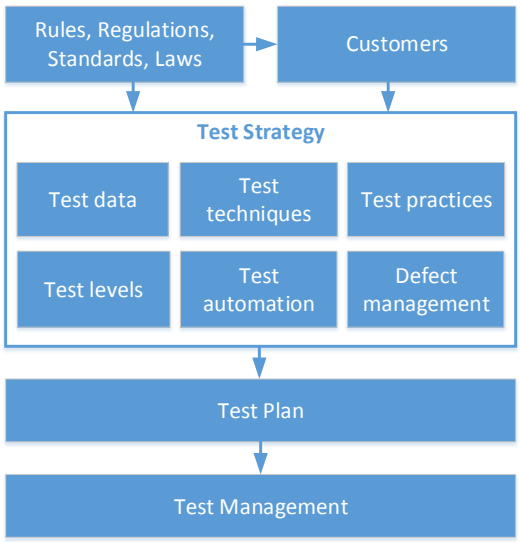


Figure 2.2 *Different factors in the testing context that test management has to take into account. Adapted from (ISO/IEEE/IEC 2013, p. 16)*

testing. (Chen et al. 2004, Walia 2012) Metrics serve as a reliable basis for predictions and decisions and as guidance for efficiency improvement efforts. Effective measurements help organizations understand their capabilities and to anticipate defects. With metrics, organisations can have better control of costs and schedule, reduce risks, and improve their product quality.

2.2 Test management

Because the objective of this thesis is identifying metrics that would support the test management of the case organisation, it is necessary to have a brief look at the activities of test management in general. In literature, the term test management is not often explicitly defined. Even when the concept of test management is discussed, two types of definitions appear. Test management has been noted to be a term that is hard to define and that can consist of many things (Gao 2011). A traditional definition of test management includes only the management of test assets, namely test cases, as test management. [1] However, a more contemporary view on test management emphasises the management of the whole testing process over the management of test cases. For example, one definition defines test management as a "broader term which encompasses different activities performed in the testing phase of a project" (Lodhi et al. 2013). Even the International Software

[1]Parveen et al. (2007) define test management as a "method of organizing test assets and artifacts such as test requirements, test cases, and test results to enable accessibility and reuse.". Craig & Jaskiel (2002, p. 186) include test case sharing and version controlling under the test management heading.

Testing Qualifications Board (ISTQB) does not explicitly define the term test management, but includes concepts such as test organization, test planning and estimation, test progress monitoring and control, configuration management, risk analysis, and incident management as being part of test management (Müller & Friedenberg 2011).

The latter, more comprehensive definition of test management is a more relevant definition in this thesis, as it captures the actual tasks in the organisation more thoroughly. Several other views on the matter were reviewed by Pohjoisvirta (2013, pp. 19-20). For this thesis, rather than analysing the formal definition of test management, it is more relevant to study the actual activities performed by the test manager and the testing team. Therefore, in this thesis, the term test management refers to all efforts targeted to facilitate testing activities in the case organization. The testing tasks performed in the organisation are described in Chapter 4.

2.3 Testing metrics, their benefits and challenges

In the literature, testing metrics are often described as part of test reporting. In other words, metrics are considered as part of testing tasks (Craig & Jaskiel 2002, Crispin & Gregory 2009, IEEE Computer Society 2008). Metrics collection should therefore be a standard activity. Measuring a test process is mentioned being a "required competence for an effective software test manager" (Chen et al. 2004). It would be difficult to manage testing without good metrics to help estimate schedules, monitor progress, make good decisions, and improve processes (Craig & Jaskiel 2002, p. 369). Most of the published test management tools, for example Quality Center by HP, Rational Quality Manager by IBM, SilkCentral, and Zephyr, support basic metrics types such as test execution data.

2.3.1 Testing metrics

Testing metrics are a subset of software metrics that focus on the quality aspects of a software development processes and products (Walia 2012). Different sources classify testing metrics into various categories. According to Black et al. (2012), testing metrics can belong to one or more of the four categories:

- Project metrics that measure the testing progress based on some established criteria or the characteristics of the project, such as cost, the number of testers, and schedule.
- Product metrics that measure some characteristic of the product, for example size and complexity.

- Process metrics that measure some aspect of the testing or development process and can be used for process improvements, for example defect removal effectiveness during development.
- People metrics that measure some aspect of individuals or groups, such as staying in schedule in test case implementation.

It is noteworthy that any metric may belong to more than one category, sometimes even to all of them. For example, the percentage of test cases executed, passed, and failed may be associated with project progress, product quality, or the effectiveness of test cases to find defects (Black et al. 2012, p. 38). Kan (2002, p. 85) would only use the first three categories, leaving the people metrics out because of their disadvantages. For example, ISTQB notes that "people metrics are particularly sensitive" (Black et al. 2012) because they can put a strain on the relationships and spirit in the organisation. In their simplest form, test metrics can be divided in two only classes: product quality metrics and process quality metrics (Walia 2012). It is debatable whether project metrics should be included in testing a metrics program or not. In general, the project metrics are not as closely associated with software quality as process and product metrics are. Nevertheless, project metrics may be utilized as normalizing factors with derived measures as they certainly affect the quality of the product. Examples include the number of testers and their skills, schedule, and the size of the project. (Kan 2002, p. 85)

Yet another way to categorise metrics is to divide them into basic metrics or computed metrics. Basic metrics (also referred as primitive or direct metrics, raw data, or fundamental measurement) are regarded as something that can be directly observed (Mills & Shingler 1988) or metrics that do not depend on any other attribute (IEEE 1998). In contrast, computed metrics (also referred as indirect or derived metrics) are calculated from basic metrics into more complex measures that provide more information (Lazic & Mastorakis 2008). Basic metrics are usually tracked in some format without calling them metrics. Examples of basic metrics include the number of test cases executed, passed, failed, or skipped, lines of code, and total development time. As testing itself is a measurement activity, product quality metrics can be seen as part of the results of the testing process (Craig & Jaskiel 2002, p. 369). In that sense, testing metrics are a required part of testing documentation. IEEE 820-2008 standard includes tracking testing status with metrics as a part of test planning tasks and the reporting of final results (IEEE Computer Society 2008).

Metrics help to gauge processes, product quality, and past performances. As a consequence, the people in the organization can more easily comprehend what is being done and what could be improved. However, almost all sources that discuss testing metrics

agree that metrics can also be easily misused or selected poorly. The risks with metrics are discussed in more detail in the next sections. Metrics benefits and risks are summarised in Figure 2.3.

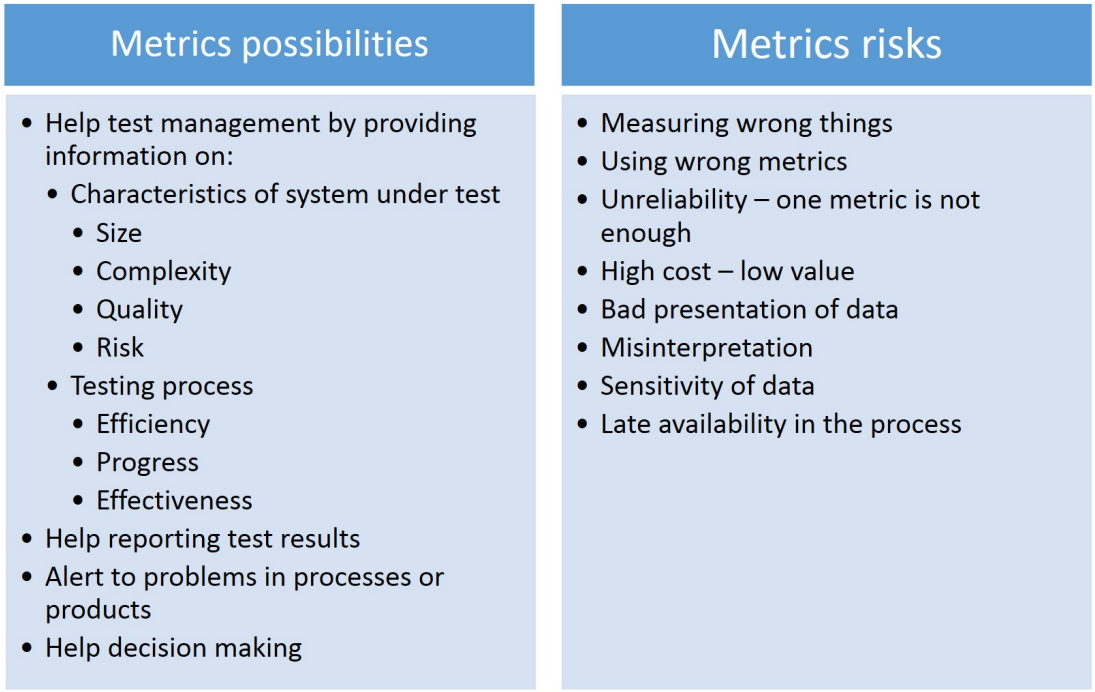


Figure 2.3 *Testing metrics have both benefits and risks. Risks should be kept in mind when using metrics.*

2.3.2 Purpose of testing metrics

One metric can be utilized for several purposes. The ways to utilize testing metrics can be classified to be useful in one or more of the following aspects: (1) planning and tracking testing progress, (2) measuring and communicating product or process quality, or (3) justified and rational decision making. The third aspect can be considered a consequence of the first two. Quality metrics may be used in establishing quality requirements, for example test acceptance criteria, and evaluating at which level the set criteria were achieved. (IEEE 1998) When metrics are used as signals to take action, they must be planned ahead of time and be based on some criteria that are based on previous measurements. That would be considered as a mature use of metrics by Craig & Jaskiel (2002, p. 373). Examples of such metrics based "flags" include test suspension criteria, pass/fail criteria for a test run, and test exit criteria.

For test managers and other stakeholders, the ability to plan testing and track its progress is vital. Metrics are a part of some quality models and standards such as the Test Maturity Model integration (TMMi) (TMMi Foundation 2015) or ISO 9001 (ISO 2008). Also,

customers from highly regulated fields may audit their potential software suppliers and request to present some basic testing metrics as a proof of quality and testing activities. Without reliable metrics, the estimation of for example how long the testing will take, how many testers are needed, and how much time and developers are needed to fix the defects rely on the test manager's experience. In an environment in which the organisation and the tested systems constantly change, experience is not enough for making good estimates. Progress can be measured based on for example the number of test cases executed and yet to run, the trends of testing pace, and the coverage of functionalities tested. Testing progress may also be measured with defect metrics, such as the number, severity, and distribution of defects in the system (Craig & Jaskiel 2002, pp. 372-373).

Most testers have understood that it is impossible to test everything even in the simplest systems (Craig & Jaskiel 2002, p. 26). It is also one of the seven testing principles of ISTQB (Müller & Friedenberg 2011). Thus, performing risk analysis is getting more and more important. (Craig & Jaskiel 2002, pp. 42-43;p. 371) For example, analysis of the trends and patterns of defects help identifying risky areas that require more testing. It has been noted that the parts of software that have had many defects in the past will likely contain them in the future as well (Craig & Jaskiel 2002, p. 140). Other factors that help finding risky areas in software include product complexity, the novelty of a feature, used technology or platform, the lack of specification, poor usability, or having a new subcontractor developing or testing the product. Knowing the most vulnerable areas in the system help planning testing to prioritise those areas over the less risky areas. It may be even more important to concentrate testing to the features and functionalities that are also important to the end users. This requires substantial knowledge of the business domain the system will operate, the end users, and the requirements.

Metrics are noted to be beneficial as they make software quality more visible (IEEE 1998). Metrics can offer reliable and, to a certain extent, comparable information about the quality of the product as well as the processes associated with the product. The quality of the product can be monitored over time, for example, from release to release to compare the performance of the organisation. On the other hand, metrics can be used to assess when the product is ready to be released in terms of quality, for example in the number of open defects. Other examples of metrics that can be used to criticize the product quality are defect severity, defect density, test coverage, or the pass rate of testing. Following patterns and using the information from past experiences may help to identify process improvement opportunities or training needs. Certain type of defects appearing might indicate training needs in that specific area or defects being inserted at a certain phase more than the others might indicate that the process can be improved or simplified. (Craig & Jaskiel 2002, pp. 372–373)

On a more daily basis, metrics can signal if the performed activities are taking the testing team in a good direction or alert the team to problems. Examples include tracking code coverage and the number of unit tests regularly, to make sure the team is not slacking on unit tests or is improving their performance towards the set goal. (Crispin & Gregory 2009, p. 75) It is noteworthy that measuring activities can have both good and bad effect on people. But when used properly, metrics can serve as a motivation. Seeing the number of unit tests go up over time is nice feedback and motivation for the teams. (Vuori 2014, Crispin & Gregory 2009, p. 76) Testing metrics may provide valuable feedback for the developers as well. The long term goal in the metrics program of an organisation should be to develop quality software. Metrics can be established to provide feedback on the organisations performance throughout the process. (Kan 2002, p. 100)

Finally, metrics may help in the process of introducing change by emphasizing the benefits of the proposed change. This might help achieving acceptance and buy-in among stakeholders, such as management, development and testing teams, or customers. These metrics may be any sort of proof in the industry, such as industry metrics or testimonials. For example, when trying to introduce new testing activities, like code inspections into the organisation, it may be beneficial to present positive experiences from other companies. (Craig & Jaskiel 2002, p. 403)

2.3.3 Challenges and limitations of testing metrics

In practice, metrics are not so straightforward but full of contradictions. Metrics established with good intentions may be terribly misused, for example by using them as a basis of evaluating individual performance (Crispin & Gregory 2009, pp. 74-75). There are also attitude issues to confront. Many software engineers regard metrics as threatening because they have little knowledge about what good metrics really can do and avoid being measured (Craig & Jaskiel 2002). A popular quote used in the field from Benjamin Disraeli: "There are three kinds of lies: lies, damned lies, and statistics.", that summarizes one key problem with metrics and highlights that testing metrics are like any other statistics (Craig & Jaskiel 2002, Crispin & Gregory 2009, p. 378; p. 77).

Metrics hold a very powerful position in organisations when they are used to determine the success of activities. While it is good to base decision making on exact and quantified information, great diligence should be taken when deciding what to measure, which metrics to use, how often to report it, and the ways to present the information. The next paragraphs summarize the disadvantages of metrics that the organisations must take into account and possibly mitigate. (Black et al. 2012, Craig & Jaskiel 2002, p. 374-381)

One metric is not enough

The most often noted feature of testing metrics is that one single metric is not enough (Chen et al. 2004, Black et al. 2012). One metric is rarely reliable enough for major decisions to be based on that only. Each metric can usually measure only one aspect of software or testing quality. Also, a single metric may not be able to separate all factors affecting the outcome. Therefore, supplementary metrics are needed to support and validate other metrics (Chen et al. 2004, Craig & Jaskiel 2002, p. 374).

For example, a test case and a defect are both very regularly utilized units in testing metrics. However, they both have the same deficiency: they are not created equal (Craig & Jaskiel 2002, p. 85; p. 258). Different test cases may differ in their code coverage significantly even when they are for the same system. Test cases for different systems are incomparable altogether. The same concepts apply to defects. Defects differ for example by their type of harm, the extent of their impact, and difficulty of fixing. Also, one fault in the system may result in multiple defects. The defect number alone would not be sufficient to imply much about the quality of a system. Things like the size of the system, the number of test cases run, code coverage, and the past trends affect the assessment. In conclusion, a combination of good metrics that measure different aspects of the system under test and the testing effort should be selected. (Black et al. 2012, Chen et al. 2004, Craig & Jaskiel 2002)

The value and cost of metrics

As any investment, testing metrics should have an adequate return on investment (ROI). This means, that the metrics should provide more value for the organisation than the effort and time they take. It can take a significant effort and time to gather and analyse the metrics. If the measurements are not used or provide no higher intelligence, they are a wasted effort. To avoid this, metrics and their data collection should be chosen carefully. It is recommended to automate as much of the data collection and analysis as possible. Recently, this has gotten easier as tool support for data collection has advanced greatly. Still, some metrics will probably have to be collected manually, for example the time used for test planning. Ideally, the metrics would be collected as a by-product of another activity (Craig & Jaskiel 2002, Crispin & Gregory 2009, pp. 375-376;p. 79). Also, the data collection interval should be adjusted to a reasonable level. The interval should be based on the needs of the stakeholders. Also, the measured object may vary too much or too little to be for it to be reported too frequently, for example daily. A week is often a more stable unit of measurement but still offers adequate granularity.

On the other hand, recently it has been acknowledged that in some cases the testers' judgement may be a more valuable "meter" than formal metrics would. This is precisely

because of the difficulties formal metrics bear. A human perception is useful especially on a higher level. Examples include evaluating whether a system has been tested properly or if the software is good enough to be released. In other words, while it is good to have metrics, testers should also be encouraged to use their intelligent mind and intuition (Vuori 2014).

Misinterpretation and sensitivity of test reports

Selecting good metrics is not enough for them to provide value. There are usually several ways to interpret one data. Because metrics are used in decision making, the risk of misinterpretation possesses a great danger. This is also a key deficiency of metrics. Metrics results must be analysed and reported in a way that they provide the required information and convey an unbiased view of the situation (Vuori 2014, Crispin & Gregory 2009, p. 78). While making interpretations and deciding on how to present the data, one should look for alternative interpretations.

Another factor that can cause misinterpretations from metrics is a phenomenon called "Hawthorne effect". It includes that even a mere process of measuring human activities affects the results (Craig & Jaskiel 2002, p. 379). Usually, Hawthorne effect is mentioned in a positive context, in which people feel motivated and perform better because managers are expressing concern and interest in them. However, Craig & Jaskiel (2002) argues that the effect depends on what is being measured. It may be that having metrics that can reveal disadvantages of the people being measured, cause them to feign results. This ultimately makes the metrics a waste of effort and even harmful if they are used for decision making. For example, setting a goal to reduce the number of defects in the system may cause developers and testers to not report all defects and find other ways to work on the issues. Or, giving the testers a goal to double the number of test cases is likely to result in a sought-after number of bad test cases. Good intends of measuring may cause people to be unproductive and unmotivated. After all, people do not often like being measured.

To mitigate the risk of misinterpretation, Craig & Jaskiel (2002, p. 378) recommend for example presenting the raw data with the processed or interpreted data to find new interpretations for the data, or make the audience reflect on the results while giving the managers hints of how their audience is thinking. Also, providing the people who are affected by the metrics adequate training may reduce the risk. Rather than looking at only the latest number of any measurement, following trends is more important. The trends and significant changes provide higher level information of whether the progress is going in a right direction or not and alarms to problems. (Crispin & Gregory 2009, p. 75)

Some data may be sensitive information to groups or individuals and managers should act accordingly with the data. Using metrics to point out developers or teams that produce

most defects has potential of doing more harm than good to relationships and spirits in the organisation. It is obvious that the managers may benefit from such information in risk analysis and in finding training and process improvements but it is recommended to weight the benefit of collecting such sensitive information. Alternatively, some metrics information may be sensitive organisationally and therefore subject to limited usage and low value. (Craig & Jaskiel 2002, p. 377)

As a summary, modelling the real world's phenomena with metrics is a standard requirement, but the implementation of good metrics is as hard as figuring out the actual concepts. Simplicity is usually recommended when starting with metrics and metrics validation should play a bigger part in the process (Chen et al. 2004). Metrics should be used but the limitations should be taken into account when analysing results or interpreting data. (Kan 2002, p. 93) It might be tempting to measure what is easy to measure, but rationale is needed when establishing metrics (Vuori 2014). At the very least, when putting metrics into use, one should be constantly aware that the metrics may not reflect the situation as it is. The process of choosing and recommendation for implementing metrics is examined in more detail in the next sections.

Availability or adequate reliability only after release

Unfortunately, many good testing metrics are available only after testing has been completed and the software has been released, such as the total number of defects in the system. This kind of metrics are also called "after-the-fact" metrics. (Craig & Jaskiel 2002, p. 277) Instead, there are metrics that try to predict the final values with the metrics that are available during the project. However, there are few in-process metrics that have been validated sufficiently. For an organisation to use a metric that has not proven its reliability in extensive experimental usage, careful validation would be necessary but is rarely done in practice. (IEEE 1998) Metrics validation is discussed in more details in Section 3.1.5.

All these thoughts are well summarised in a few sentences by H. Thomas Johnson: "Perhaps what you measure is what you get. More likely, what you measure is all you get. What you don't (or can't) measure is lost." (Johnson 2006)

3. DEVELOPING TESTING METRICS IN AN ORGANISATION

There is a strong agreement in the literature, that there are no specific metrics that should be used. This can be concluded from the various suggestions for the process of choosing metrics, for example by Craig & Jaskiel (2002, p. 382) and Crispin & Gregory (2009, p. 76-77). Also, there exist more than one approaches that can be used to plan metrics programs. The first is the Goal/Question/Metric (GQM) paradigm developed by Basili & Weiss (1984), and the second is the Software Quality Metrics Methodology standard published by IEEE (1998). However, IEEE, too, highlights that the standard is a process standard, not a standard that dictates specific metrics for organisations to use. This is because each organisation has different characteristics, requirements, products, projects, and processes. (IEEE 1998) While both approaches have been accepted by the industry (Mills & Shingler 1988, Kan 2002) and are relatively similar, the IEEE standard was chosen for a closer examination in this thesis because it is more structured. After an introduction to the IEEE standard, a selection of often mentioned metrics in the literature are introduced in Section 3.2.

3.1 Process for selecting metrics

The IEEE standard introduces five high level steps for establishing software quality metrics programs. It is also recommended to periodically review the need for each metric (Craig & Jaskiel 2002, p. 376). Therefore, metrics development should be an iterative process in which the metrics set is continuously improved and validated. The steps of the metrics development process is visualised in Figure 3.1 that highlights its iterative and continuous nature.

Several other sources often emphasise some parts of the process or discuss the steps with different terms (Basili & Weiss 1984, Craig & Jaskiel 2002, Crispin & Gregory 2009). Still, the other approaches to the development of software metrics essentially support the IEEE standard. In addition, the literature offers numerous basic principles for metrics that organisations should at least have. They are often lists of quality aspects or categories of measurement that should be measured. The Software Engineering Institute (SEI), for

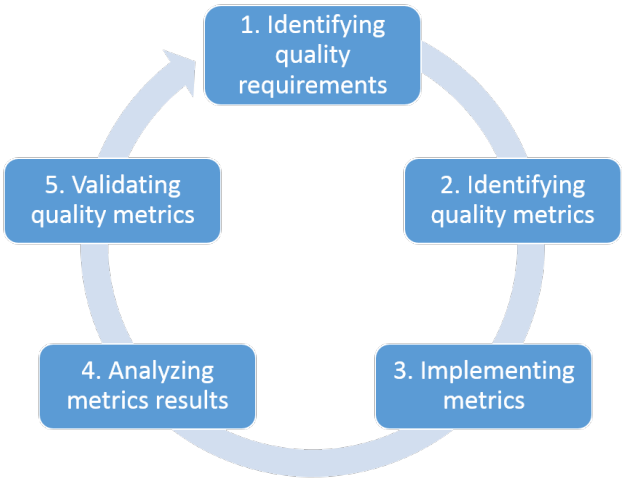


Figure 3.1 *There are five steps in the IEEE process standard for developing metrics in an organisation (IEEE 1998). As it is recommended to periodically review the current metrics set, the process becomes an iterative cycle.*

example, has proposed that there are four core metrics areas: (1) Schedule, (2) Quality, (3) Resources, (4) and Size. Craig & Jaskiel (2002, p. 382) suggest that each organisation would need at least one metric from each of the four mentioned categories. Kan (2002) presents the following testing related dimensions to be assessed to get an adequate picture of the quality of the product: (1) total number of defects, (2) remaining number of critical problems, (3) feedback from pre-release customer testing (for example beta testing), and (4) other quality attributes such as performance and security.

As a summary, choosing metrics should be driven by the goals of measurement and improvement. Simplicity, and balanced selection of metrics, careful definition of metrics, and training of people is imperative. The metrics set must be supported by people affected by the metrics. All this is required for the metrics to be effective. The drawbacks and the chance of misinterpretation must be kept in mind when analysing the metrics results. Next, the IEEE process is described step by step in more detail while also including the recommendations from other sources.

3.1.1 Identifying requirements

Almost all sources emphasise the importance of metrics goals over the metrics themselves (Craig & Jaskiel 2002, Crispin & Gregory 2009, Lazic & Mastorakis 2008, Jones 2012). When trying to figure out what to measure, the first step is therefore to understand the underlying problems the organisation wants to solve and the requirements the organisation needs to meet (Crispin & Gregory 2009, p. 76). Only after the problems are known, the goals can be set. The goals may concern internal or external requirements such as laws,

standards, regulations, third party requirements, budget, or process control (IEEE 1998). The collection of requirements can first start with generic goals that are then refined. Each goal or requirement should be rated based on their importance.

In the process of defining the goals, it is recommended to involve all parties that are affected by the metrics. Especially the ones that will collect, analyse, and use the metrics and those who are being measured should be included. Involvement will not only provide a broader view on the measurement goals, but will also help gaining approval for metrics within the organisation. Finally, when the goals have been identified, each goal should be assigned one or more quantifiable measure to represent them. In other words, the quality goals should be converted to measurable factors, for example the goal to improve the testing efficiency could be quantified by phasing the goal as "increase the mean number of test cases run in a week by 10 %". (IEEE 1998, Crispin & Gregory 2009, p. 76)

3.1.2 Identifying metrics

When the measurable factors have been determined, it might seem relatively straightforward to find a metric suitable for the goal. Yet there are a great number of aspects to take into account, such as the metrics collection methods, the cost of metrics, motivational aspect and other limitations of metrics, and the validity of metrics. For each goal, there usually exists a selection of possible metrics to choose from. Thus, the selection process can be far from easy.

When choosing the metrics, simplicity helps to avoid many problems right from the beginning and should be pursued especially if the organisation is new to metrics (Chen et al. 2004, Lazic & Mastorakis 2008). It is also noted that the first task of metrics is to answer the question of whether the testing activities are doing "the right thing" (effectiveness). Then, once there is assurance of the quality of testing, metrics can be created to measure the efficiency of testing. (Lazic & Mastorakis 2008)

Metrics validation and measurement quality are topics that are only sometimes brought up when testing metrics are discussed in the literature. Ideally, each newly created metric would be formally evaluated at a metrics research center, for example at a university, or a non-profit research facility. In practice, however, this does not happen and new metrics are frequently introduced in research papers without formal evaluation or experimental proof in controlled environment. (IEEE 1998, Kaner & Bond 2004) As basic metrics are presumed valid and usable even without further validation, there is another reason to start metrics programs with simple basic metrics (Lazic & Mastorakis 2008).

Even if it is said that each organisation has its own characteristics to which the metrics should be based on, it is acceptable to use the metrics that some other organisation has been using as a starting point. However, one should prefer organisations that are well established in the industry and have proved the metrics' usefulness through extensive experimental usage (Jones 2012, Kan 2002, p. 302). Metrics that have been heavily used in practice and in various environments are regarded more reliable. For example, Craig & Jaskiel (2002, p. 431) argue that a test manager should measure coverage and have at least one other metric for test effectiveness such as Defect Removal Efficiency (DRE). Kan (2002) in turn highlights the importance of customer satisfaction metrics as it is also good practice to consider the customer's perspective even if the main quality improvement effort should come from the development. His suggestions for metrics programs with minimum resources include (1) defect arrival metric, (2) number of defects reported after release, and (3) normalising of computed metrics with product size metrics such as lines of code (LOC) or function points (FP). Kan (2002) has compiled examples of metrics programs from organisations such as Motorola, Hewlett-Packard, and IBM Rochester that can be used as a reference. Using the best practices in the industry can justify the metrics program, for example to auditors. In contrast, using some newly published metric without extensive validation will raise suspicions for the metrics reliability. Nevertheless, it is important to evaluate the quality of the information being reported. Validation of metrics is described in more details in Section 3.1.5.

As discussed in Section 2.3.2, risk analysis and risk-based testing approaches have become increasingly popular in software quality assurance and most organisations do risk analysis in some format. Metrics should be planned to also support this risk-based approach. For example, metrics like defect density or relative complexity of a feature can be used to determine the likelihood of error in different features (Craig & Jaskiel 2002, p. 39). For risk management itself, there are performance metrics available, such as the risk growth performance index and the risk cost performance index. They can be utilized when the metrics program and risk management processes of the organisation are on a mature level (Lazic & Mastorakis 2008).

Sometimes the structure of used processes are in themselves very difficult to measure. Using existing measurement frameworks as a reference may help finding suitable metrics. For instance, the effectiveness of exploratory testing is challenging to measure as the only concrete output may be defect reports and little evidence of the time to find them. Therefore, reporting the status of testing and justifying the adequacy or inadequacy of testing to the stakeholders may be very difficult. (Bach 2003) Also, the effectiveness of exploratory testing depends on so many intangibles that are difficult to quantify, such as the skill of the testers, their intuition, creativity, experience and ability to recognise productive areas. To overcome this, more formalised processes for exploratory testing

and measuring it have been developed, such as Session-Based Test Management first introduced by Johathan Bach in 2000. It includes a selection of metrics that measure the time spent on certain tasks during the testing, the number of issues found, and the functions covered during the testing time. (Bach 2015)

Finally, when the metrics selection has been refined, the organisations need to set the desirable limits or the targets for the metrics. The true value of metrics should be to be able to drive improvements. Each metric should indicate "good" or "bad" and therefore, a comparison scale should be set up. (Kan 2002)

3.1.3 Implementing metrics and reporting results

It is said that the actual implementation of metrics is one of the hardest parts to overcome when establishing a metrics program. Even with many good metrics defined and the organisations quite aware of the importance of metrics as part of best practices, the adaptation level has been quite low. In 1999, the mean adoption level of metrics was 45 percent within almost 400 companies in Europe (Dutta et al. 1999). Chen et al. (2004) and Craig & Jaskiel (2002, p. 381) identify the lack of training as one of the problems in adopting metrics. According to them, the first task in implementing metrics is to ensure that everyone affected by a metric support and understand the value of metrics. This includes those collecting metrics data, those being measured, and those making decisions based on them. Training should include at least the reasons for collecting the metric, the decisions they can affect, principles of metrics interpretation, as well as what each individual can do to affect each metric positively. Training is also important to ensure the data is collected and analysed in a consistent manner. This is because it has been observed that people tend to explain away the negative signs indicated by the metrics (Kan 2002, p. 323).

At this point of a metrics development process, it is recommended to establish a traceability matrix or a metrics data sheet to map collected data items and metrics. The goal of a metrics data sheet is to help understand the collection and purpose of each data item and metric. It also helps in making the metrics collection and analysis more consistent and provides a tool for periodically reviewing the value of each metric. (IEEE 1998, Craig & Jaskiel 2002)

After identifying a set of needed quality metrics, procedures and possible tools for collecting raw data should be defined. This includes determining the data to be collected, by whom, when, and how the data is collected and where it is stored. (Crispin & Gregory 2009, p. 79). Then, the methods of analysing and presenting the raw data should be

defined. While defining these details, organisations should keep in mind all the recommendations and risks with metrics that were discussed in Section 2.3.

Many sources recommend prototyping the measurement process with sample data or piloting the metrics program in a small project before gradually deploying the metrics across the whole organisation. (IEEE 1998, Chen et al. 2004, Kan 2002) By piloting, metrics program and all instructions concerning data collection and analysis can be checked to ensure that the selected metrics provide reliable results and improved if necessary. Eventually, the metrics program can be started in the whole organisation. The collected data should be checked for accuracy and uniformity on a regular basis in the beginning. (IEEE 1998)

3.1.4 Analysing metrics results

This phase includes interpreting the results of software metrics and making conclusions about the software quality, its compliance with requirements, and identifying process improvements. It is recommended to analyse metrics results on regular intervals during the testing and particularly after the system is released. The measurement results can be compared with the targeted values and previous measurements. Comparisons with other projects, releases or the measurements of some other organisation may be done to determine if the results are typical in the industry. However, caution is needed when comparing dissimilar objects. Usually, the trend is more important than a snapshot of a value at a specific time (Crispin & Gregory 2009, p. 358). Significant differences or recognised trends should be further investigated in order to find the root causes. Looking at all available data and several different metrics in conjunction may give a more complete picture of what is really happening and help in the root cause analysis. (IEEE 1998, Lazic & Mastorakis 2008)

For example, the organisation may assess the software quality based on metrics result by comparing the measured values to the established objectives and tolerance limits. If unacceptably low quality is observed, it may be attributed to excessive complexity, poor documentation, or bad design. Based on the results of the analysis, organisation may decide their course of action. Usually, the decision is made between releasing the product as it is or delaying the release to improve the product quality. In a very bad situation, the decision may be to abandon the whole project and look for another solution. (IEEE 1998) Other decisions may be made, such as to flag a certain software component, a process phase, or a development team. for further investigation or follow up (Lazic & Mastorakis 2008).

As stated previously, organisations are encouraged to re-evaluate their metrics program regularly, for example annually. Crispin & Gregory (2009, p. 366) notes that release

planning, for example, is a great phase to evaluate the Return on Investment (ROI) of the metrics that have been collected and consider if the metrics program could be improved for them to provide more value, for example by automating some manual data collection. It should be evaluated whether the metrics really are collected correctly, how much effort the data collection takes, whether the metrics are guiding the development and testing effort to the right directions, and if they really provide valuable information.

3.1.5 Validating metrics

Metrics, as any statistics, need to be validated to prove that a statistically significant relationship between a set of metrics and the quality factor exists. Metrics validation aims to ensure that the metrics can accurately predict and quantify the state of the interesting quality factors. It is usually assumed that direct metrics are valid by nature. However, many direct measurements can be done only after release or later in the project, such as the measures of reliability. Therefore, computed metrics are used to predict those quality factor values during the projects. These computed metrics need to be validated. (IEEE 1998)

IEEE Standard for a Software Quality Metrics Methodology (IEEE 1998) notes that validation of metrics does not refer to a universal validation of the metrics for all applications but only in a specific application. Examples of software quality metrics' validity criterion include correlation, tracking, consistency, predictability, and discriminative power. According to IEEE (1998), the validation procedure consists of the following steps: (1) identifying a sample of the quality factor data, (2) identifying a metrics sample from the same domain as in step 1, (3) performing a statistical analysis, (4) documenting the results, (5) revalidating the metrics, and (6) evaluating the stability of the environment. Validation of metrics is not part of the scope of this thesis and therefore, the validation procedure steps are not examined any further.

3.2 Popular metrics in the field

The literature was reviewed in order to have insight into what kind of metrics are recommended or used in the software testing field. This was done with an objective of getting ideas of what metrics to develop in the case organisation. In this section, various metrics identified from the literature are described and evaluated. Testing metrics were identified from academic literature, from other organisations in the field, and from various available testing tools. The found metrics are summarised in Figure 3.2. Usually in the literature, metrics descriptions are not detailed enough for the metrics to be straightforwardly adopted into an organisation. (Lincke et al. 2008).

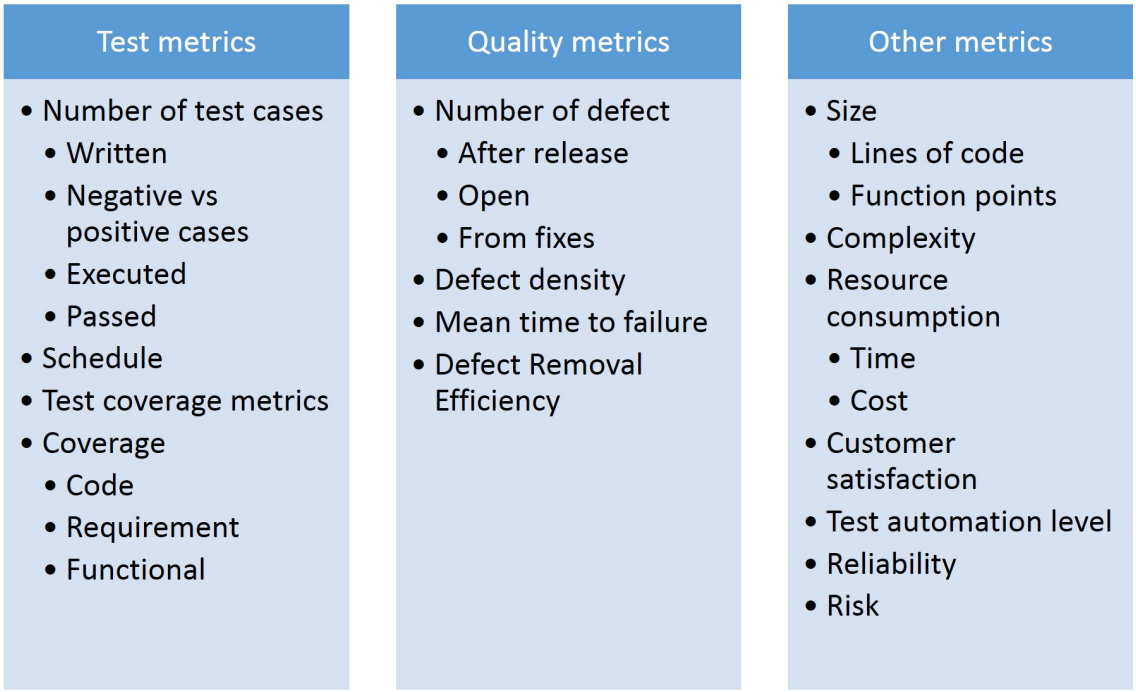


Figure 3.2 *A summary of metrics often mentioned in the literature. List compiled from (Kan 2002, Craig & Jaskiel 2002, Chen et al. 2004, Kivimäki 2007, Lazic & Mastorakis 2008, Jones 2008, Crispin & Gregory 2009, Müller & Friedenberg 2011, Jones 2012, Black et al. 2012, Walia 2012, Vuori 2014).*

3.2.1 Test metrics

Measures from test cases and their execution are one of the simplest and frequently mentioned metric in the literature. However, increasingly, they have been criticised for being unreliable as the quality of test cases may vary greatly and the number can be easily manipulated for example by creating duplicate or trivial test cases that bring no value.

The number of test cases

The *number of test cases* is a very basic and easy metric many authors mention at least in some format (Müller & Friedenberg 2011, Lazic & Mastorakis 2008, Crispin & Gregory 2009). A test case number is often easy to gather automatically with proper tool support of a test case warehouse. This metric requires some sort of classification to be useful. The bare number without any context or comparisons is only that, a number, and holds no additional insight into the test cases. Number of test cases may be reported for example by a feature, a component or a requirement of the system, by a release, by a tester, against time, and many more. The central problem with test cases is that they are not equal with each other in size or coverage (Craig & Jaskiel 2002, p. 85; p. 258). There are also different types of test cases, positive and negative. This disadvantage can be offset by weighting test cases with a size measure. The size measure may be for example time

units, or some type of coverage unit (Craig & Jaskiel 2002, p. 259). When adding some other details to test case number, it may help in estimating for example the test coverage, required testing time, and quality of testing.

Test case execution data

An enhanced version of the number of test cases is measuring the *number of test cases executed*, passed, failed, skipped, and others. Test execution data is often more versatile than a plain number of test cases, especially when combined with testing time measurement. In addition to the test case classification, test execution data can be derived to more advanced metrics such as *pass rate*, *fail rate*, *test execution pace*, and more. Test run rate (the test cases executed out of all planned test cases) gives insight into testing progress. Together with testing time, test execution rate can help predict when the testing will be ready. Test pass or fail rate helps in evaluating the quality of the system under test. (Kivimäki 2007, Craig & Jaskiel 2002, Kan 2002, Lazic & Mastorakis 2008, Garrett 2011)

Test coverage metrics

As mentioned earlier, test cases test, in other words cover, a certain part of the system. Each test covers a different sized part of the system and usually the areas overlay, more or less. Test coverage is a traditional metrics and there are several open and commercial tools available to automate it.

Code coverage is regarded as one of the most powerful measures of test effectiveness because it can be utilized during the project, and not only after testing is done. There are different types of coverage measures, such as statement coverage, decision or branch coverage, and path coverage. Code coverage is said to be the only metrics that can give assurance that the software is completely tested. It can also highlight the parts of the software that are not tested. (Craig & Jaskiel 2002, p. 181) However, when interpreting code coverage, it should be noted that code coverage measure tells only how much of the code was run during the tests. It does not take account the quality of the test cases or if the code was doing what it was supposed to do. It is recommended to use code coverage in the low level testing for example as an indicator that encourages and reminds to do unit testing, or to measure how much of the functionality is covered by test automation. Also, purely striving for high code coverage is not recommended as testing should not only aim for executing the code but also testing if it is doing what it should. Rather than that, a mature goal of using coverage metrics would be to achieve high code coverage as a side effect of testing. (Crispin & Gregory 2009, p. 358)

Coverage can be measured on a higher abstraction level as well. Examples of higher level coverages include *requirement coverage*, *user story coverage*, and *risk coverage*

(Vuori 2014, Garrett 2011, Black et al. 2012, Kivimäki 2007). These metrics may help to measure and guide exploratory testing effort by allowing to quantify the quality and amount of effort put into testing. (Bach 2015) Coverage metrics may be used when determining the desired coverage levels based on risk analysis. They can be useful also for communicating the testing level objectives to the business stakeholders that do not necessarily understand the concept of test cases. (Black et al. 2012)

3.2.2 Quality metrics

Other very basic metrics are the defect metrics. Together with the test case metrics, they form the minimum set of metrics an organisation might need.

The number of defects

Like test cases, one can measure the plain *number of defects*, or classify the defects by component, by feature, by severity, when they were introduced to the system, who they were reported by, when they were reported, whether they are fixed, and many more. However, like test cases, defects are also not equal with each other and need some sort of normalisation. The number of open defects and their severity is one way to assess whether the system is ready to be released in terms of quality. The trend in the number of defects over time helps to estimate the number of expected defects in the future testing tasks. Changes in the trend on the other hand should alert the organisation to problems. (Craig & Jaskiel 2002, p. 266)

The number of found defects can be used to evaluate the effectiveness of testing to find defects. However, the number of defects found in testing depends also on the initial quality of the system. This effect is difficult to take into account as it is impossible to know how many defects were not discovered in testing. The number of defects reported by end users can be used to estimate the remaining defects. However, this information is not available until after the release and therefore not helpful during the testing. Metrics that utilize the customer's input are described in more detail shortly. There is also the problem of how to take into account the defects found during the development, or if to take them into account at all. Still, some organisations use the number of found defects in comparing different testing methods. This requires consistent testing practices and well established test sets, good defect tracking and analysis methods. (Craig & Jaskiel 2002, p. 273)

Defect density

As noted earlier, testers have often experienced that the areas of the system that have had a lot of defects in the past are likely to have them in the future as well. The number of defects found during previous testing efforts can be utilized in directing testing effort into the most error prone areas. This type of risk analysis based on defects is typical in many testing organisations. Its purpose is to help defining the likelihood of failure. An example of a metric that quantifies the risk in different areas of the system is *defect density*, also known as defect potential. Defect density is defined as the number of defects in some measure of system size, such as lines of code or function points (described in more detail in Section 3.2.3). Unfortunately, defect density metrics have several issues. The problems lie in defining how to count defects and how to quantify system size. (Lazic & Mastorakis 2008, Kan 2002, Craig & Jaskiel 2002, p. 284)

Lazic & Mastorakis (2008) highlights two software quality metrics as critical to the industry: *defect potential* and *defect removal efficiency* (DRE). Defect potential, i.e. defect density, was discussed above. Defect removal efficiency is a metric that is mentioned in most testing metrics sources over the years (Kan 2002, Craig & Jaskiel 2002, Chen et al. 2004, Lazic & Mastorakis 2008, Garrett 2011, Jones 2012). It is recommended as a powerful and useful measure for test effectiveness by both Craig & Jaskiel (2002, p. 276) and Jones (2012), though not without issues. DRE is defined as:

$$\text{DRE} = \frac{\text{Number of defects found in testing}}{\text{Number of defects found in testing} + \text{Number not found}}. \tag{3.1}$$

DRE represents the percent of defects that were found out of defects that could have been found. It can be calculated to different test phases or the testing effort overall. When calculated for a specific test phase, the numerator becomes the number of defects discovered during a specific phase and denominator is the number of defects found during the phase plus the defects found after the specific phase. Usually, the number of defects not found is the number of defects found by the customer. In 2007, the mean defect removal efficiency in the U.S. was about 85 percent (Jones 2008, as cited by Lazic & Mastorakis 2008)

As mentioner before, not even DRE is without challenges. Firstly, it is another after-the-fact metric that is not available during actual testing and can only be used for retrospective. Secondly, it has the same issues that defect count measures have: defects are not created equal and it is impossible to know when all defects have been found. The shorter the time for customer usage in calculating DRE, the higher the DRE values are, because more defects not found in the testing are left out from the denominator. The longer the

customer usage time is, the more realistic the DRE value is (Jones 2012). According to Kan (2002, p. 88), most of the defects in software systems are usually found in two years after release. However, two years is too long of a period for the measurement to have any actual value. The literature has recommended a post-release interval of 30 or 90 days. As a 30-day time span is arguably too short for end users to even properly start using new software, a 90-day time period would be better for judging the quality of software. (Jones 2014)

A general approach is also to use the information from the issues reported by a customer to estimate the time it takes for the customer to find most of the defects. On the other hand, if a defect is not ever found, it can be considered as a defect that can not be found regardless. However, only a fraction of defects found by the customer get reported, even if their severity is high. The users may not bother or know how to report the issues or just drop the software from use. With several customers, there is also a problem that defects severity is not the same for all customers. This makes weighting the defects difficult. Other difficulties lie in defining when to start counting the defects, whether to only count the defects found by the testing team, and whether to count the defects that could not have been found in testing no matter what due to, for example environmental limitations. However, more important than trying to find the best definition for each detail is consistency over time. Otherwise, comparison between measurements is not reliable.

All in all, DRE is noted to be a simple and an inexpensive metric that has the ability to capture the quality of the project and the software on a single figure. It also suffers only from a minimum amount of problems that defect count measures have, as reviewed above. (Jones 2012)

3.2.3 Other metrics

This section includes some other metrics that can be used to measure the system size, its quality, the testing effort, or test automation.

System size and complexity metrics

Other general aspects that are often measured in a system include its size and complexity. For system size, the two most common measures are *lines of code* (LOC) and *function points* (FP). These metrics are usually used for normalisation of other metrics, for example when examining coding speed, complexity, test coverage, and many more. LOC is a very simple metric and several tools that compute the number automatically are available. It has a disadvantage of being highly dependant on the programming language and subject to programming style and other quality factors. For example, the cost for a single line

of code is higher for higher level programming languages than for low level languages like assembly language because a high-level language code usually has fewer lines. On the other hand, the code could be squeezed to a very few lines of code while giving up maintainability. Despite these issues, LOC may be the best option due to its simplicity and extensive tool support when the problematic variables can be assumed constant for example in subsequent releases of a product in the same development organisation. (Jones 2012, Kan 2002, p. 101)

The alternative to measuring software size is the function point which is used to illustrate the size of a functional requirement. The rules for calculating function points are currently issued by The International Function Point Users Group (IFPUG). Similar variants used in the industry are for example *use-case points* and *story points*. Function point metric is viewed preferable over LOC because it does not depend on the programming language or coding style and quality. Rather than that, its purpose is to measure the size of functionality, not the implementation. It is also available even before the implementation takes place, in contrast with LOC. However, function points have problems too. The metric is criticised for its high cost and laborious manual counting of function points. (Jones 2012, Kan 2002, p. 94) Also, Albrecht & Gaffney (1983) observed in his studies that function points are correlated to lines of code, which makes it questionable whether it is equivalent with lines of code, except enhanced with additional problems. This observation may be outdated as the methods of counting function points have been developed. (Jones 2012)

Cyclomatic complexity is another widely discussed complexity measure. Cyclomatic complexity is defined as the measure of the branches in the program control flow. A perfectly structured program with no branching would have the cyclomatic complexity of 1. Cyclomatic complexity measure can be used as an indicator of excessive complexity or to evaluate the amount of required test effort, for example in a module. (Jones 1994, Kan 2002)

Resource consumption

Resource consumption data, examples including *testing time*, *staffing level* or *machine time*, is important information for test management when planning future testing efforts and monitoring testing progress. Time is a simple measure, but it is usually difficult to define how much time a single task takes because there are always interruptions and problems when working. Therefore, whenever possible, calendar time or business days should be preferred over effective working hours. (Craig & Jaskiel 2002, Chen et al. 2004, Kivimäki 2007, Walia 2012, Müller & Friedenberg 2011)

Customer satisfaction measures

Customer satisfaction is generally measured by analysing help desk calls, defects reported by the customers, and customer satisfaction surveys. Each method has its problems: defect measures have the usual problems characteristic of them while good surveys are very difficult to create. Creating surveys is a whole discipline of its own and not examined any further in this thesis. (Kan 2002, Craig & Jaskiel 2002)

It is considered good practice in software quality engineering to also consider the customer's perspective. It is important because no matter how development teams perceive the product quality, it is customer satisfaction that defines the success and future of the product. For a customer, the total number of defects that affect their business is more important than any other measure. However, a problem that is serious for one customer may not be that to another (Chen et al. 2004). Because different customers may have different priorities for functionalities, it is sometimes difficult the evaluate the customer satisfaction. Measuring and using customer satisfaction measures to assess the quality of a software and testing effort can be therefore considered a mature way of using metrics. (Kan 2002, p. 92)

Test automation metrics

Test automation metrics are metrics that measure the performance of the implemented automated testing process. They are used to help improve the test automation processes of the organisation and tracking its status. Test automation metrics are important especially when starting test automation and evaluating whether it is beneficial to invest in. The questions are, in other words, what is the cost of developing test automation, how much resources and time it saves, and its effectiveness in finding defects. The cost of maintaining test automation, for example updating test cases and researching possible problems, should also be taken into account. Possible metrics for test automation include measures such as the number and percent of automated test cases, the number and percent of test cases that can be automated, amount of resources required for developing the automated test cases, amount of resources required for an equivalent manual testing effort, and the number of defects found by test automation. (Garrett 2011)

4. TESTING, TEST MANAGEMENT, AND TEST METRICS IN M-FILES

Software metrics and models do not exist in a vacuum. They must be planned in the context of the software development process (Kan 2002). There are various different software development process models used in the software development community. Therefore, it is essential to look at the process of an organization that metrics are being developed for. This chapter introduces M-Files as an organisation and their main product. First, the product development process and the testing tasks during the development process in M-Files are described. Then, an overview of the testing practices and used tools in test asset management, defect management, and test management is given. Finally, the problems in testing practices and current metrics are assessed.

4.1 M-Files and its product development

M-Files Corporation, previously called Motive Systems, is a software company founded and based in Finland. The company employees more than 250 people and has offices in Tampere and Espoo in Finland, and Dallas in the United States. Their main product, that carries the same name as the company, is an enterprise content management (ECM) system based on content metadata. Its competitors include for example SharePoint and OpenText. In M-Files, all documents, files, and all other information is saved as various types of objects to so called "vaults". Each vault's object structure and features can be defined with M-Files Admin tool. The system has numerous features, such as version history, workflows, electronic signatures, user permissions, PDF conversion. It also integrates with several other products, for example Microsoft Office variants and AutoCAD. A screenshot of M-Files user interface is in Figure 4.1. The system can be deployed on-premise, in cloud, and in a hybrid environment. The product is fully available only in Windows environments, but separate clients are developed for a limited access via web browsers and mobile devices. Additionally, M-Files offers product variants, such as the M-Files QMS (Quality Management System), and allows further customisation with application programming interfaces (API) and scripts. (M-Files Corporation 2015a)

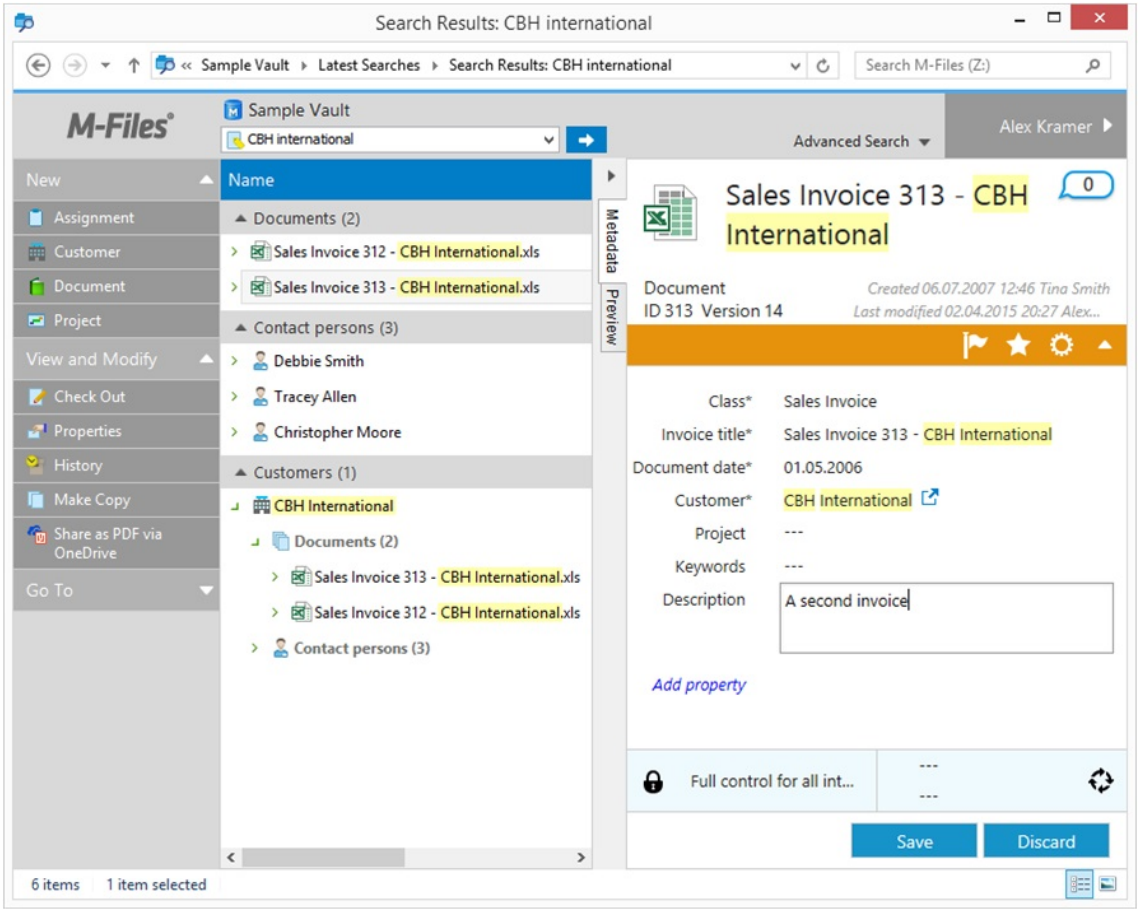


Figure 4.1 *M-Files user interface with a task pane on the left, a list of search results in the center, and metadata card on the right showing the selected documents metadata.*

The main release of M-Files has an approximately annual release cycle that consists of the following phases: release planning, feasibility studies and UI concepts, implementation and testing, and finally, finalization, system testing and releasing. These phases usually overlap to some extent. Each phase ends with a major milestone which is defined for each release specifically. The release phases are illustrated in Figure 4.2. Minor releases and possible service releases are scheduled on a need basis and may be customer specific. Each new major and minor release may include enhancements and fixes in the existing features, or completely new features. The M-Files product variants and mobile applications have their own release cycles, quite separate from the main product. (M-Files Corporation 2013)

Figure 4.2 *M-Files release cycle phases that usually overlap to some extent. Each phase ends with a major milestone.*

The Research and Development (R&D) department, including the testing team, is based mainly in Tampere. At the time of this study, the department employs about 30 developers and 10 testers internally and about 15 developers and 5 testers externally as subcontractors. The department is divided into seven development teams and one testing team. In product development, an iterative Scrum-method is applied as follows: for each development team, there is a prioritised backlog of work items called user stories that are implemented in two week long periods, called sprints. Before each sprint, the priority order of the user stories is determined, and possible design and development changes are made. Each development team holds a planning meeting at the beginning of a sprint and decides which user stories they can implement during the sprint. Testing is also a mandatory part of a user story unless it can be justified unnecessary. Then, at the end of the sprint, there is a review meeting where each user story is checked against a Definition of Done (DoD), demonstrated, and accepted by the product owner unless deviations were found. The DoD check is a major quality assurance control measure in which, for example the testing and bug fixing are assured to be done. (M-Files Corporation 2013)

4.2 Testing process and test management

In the release cycle and implementation process description, the two major testing tasks in M-Files were introduced: system testing at the end of the release cycle and user story testing during the implementation phase. In the system testing phase, the objective is to retest all existing test cases from previous releases. Also, for each major release, performance testing is executed with an internally developed tool called Usage Simulator and a specific validation testing set is executed. Additionally, an extensive smoke test is run for each release, including major releases, milestone releases (Alpha, Beta, and Release Candidate releases), minor releases, service releases, product variant releases, and mobile application releases. Occasionally, the testing team has also participated in major customer projects by testing the implementation of a customised solution. User story testing includes planning and writing tests, executing the tests, and reporting the results and found defects. In system testing, the selected tests are re-run again, and the results are reported. Test asset and defect management are explained in more detail in the next sections. (M-Files Corporation 2014c)

This type of system testing is called maintenance testing which includes regression testing to existing parts of the system in addition to testing new and changed functionality (Müller & Friedenberg 2011, p. 30). The latest major version, M-Files 2015, which is the 11th major version, was released during this study, in April 2015. The first version of M-Files was released about ten years ago and the software has naturally evolved enormously. System testing has become challenging as the number of test cases and features

have increased, test specifications have gotten partially out of date, the system still is continuously evolving, and there are no resources to test everything at once. Usually, the scope and the level of maintenance testing depend on the risk and size of made changes and the size of the existing system. Therefore, analysing how much the existing system may have been affected by the changes should be done to determine the scope of regression testing. This type of analysis is called impact analysis and is used in the target organisation as well. (Müller & Friedenberg 2011)

The organisation has automated testing in the form of NUnit tests that exploit the M-Files server and client API and can be considered system testing in a way of continuous integration. NUnit is a free, open source unit-testing framework for all .Net languages, for example C# (NUnit.org 2015). The organisation has also developed UI test automation for both the desktop and the web client of M-Files. Test automation for M-Files mobile clients has been planned. NUnit tests are integrated into the build management and continuous integration server of the organisatin that executes the unit tests for every build. NUnit test cases are developed for new functionality based on how critical the functionality is for the end user. The UI test automation tests are used as regression testing effort and are executed for M-Files builds on regular basis and for each release build. The automated test specifications are decided based on how critical the functionality is to the end user, how long it takes to run the test cases manually, how easy is is to automate the test case, and how many times the tests are expected to be run. (M-Files Corporation 2014c)

Test management in M-Files is mainly performed by the test manager. During the implementation phase of product development, test management includes tasks such as prioritising user story testing tasks and maintaining the test assets, such as test cases and test automation. During the system testing phase, test management includes mainly planning, prioritising, and scheduling system testing, but also prioritising and planning user story testing tasks. Other possible testing tasks such as release smoke testing and customer project testing are prioritised and scheduled among other tasks. Additionally, monitoring, possible rescheduling, and reporting of the testing activities and results are part of test management throughout the development process. The next sections describe how these tasks are performed in more detail.

4.2.1 Test management and defect tracking

Recently, the case organisation took a new internally developed test management tool into operation. The tool is an M-Files vault, with an additional application that is built on M-Files platform using its development and customisation capabilities. The tool was decided to be developed in order to improve the test case management and to aid test planning,

monitoring, and reporting as described in (Pohjoisvirta 2013). Defects and improvement suggestions are stored and managed in another M-Files vault, called "Tracker". Defect reports can be linked to and accessed from test cases in the test management tool. All new defects are evaluated, assigned a priority, and scheduled for fixing by assigned pre-evaluators, such as development team managers and the senior director of R&D. (M-Files Corporation 2014b) The four main benefits of using M-Files content management system in test management and defect tracking are that it allows easy customisation of the object structure, it tracks object change history, it allows relationships between objects to be easily formed and utilized, and it has an advanced workflow feature for managing object states.

The structure of the test management tool content is visualised in Figure 4.3. The test management tool consists of *test case* objects that can be complied into *test sets* and further into *test set collections*. Test cases and sets can be mapped to belong to one or more *features*. The test cases and sets are usually created during user story testing. Then, each time a test set is run, a *test set run* object is created from the test set and the test cases that belong to that test set, representing unique instances of executing the tests. A test set run object can be created for a single test set, or a collection of them. Test run and set objects include the test execution environment details. Test run results (pass, fail, or skipped) are also recorded into the test run objects. Having the test cases and test runs as separate objects enables tracking the change and execution history of a test case accurately.

Finally, the test run objects are linked to a *test context* object which is further linked to a test plan. Usually, the test context is a user story or a system test phase, and includes information about the context the tests are being run, such as the developer of a user story. A test plan can be the test plan of a specific release, for example the system testing of M-Files 2015, or M-files 10.2. In other words, test runs are always ultimately linked to a test plan. An example test run object hierarchy is in Figure 4.4.

4.2.2 Test planning and monitoring

The test planning is mainly done by the test manager. User story testing tasks are prioritized and assigned to testers based on the importance of the changed feature for the end users and a subjective evaluation of risks involved in the changes. The developers, the product owner, or the senior director of R&D sometimes participate in this prioritisation process. The time needed for the testing task is approximated based on the previous experiences of existing features, the size of the change, the number of existing test cases related to the feature, and intuition. Usually, when a tester has finished testing a test set,

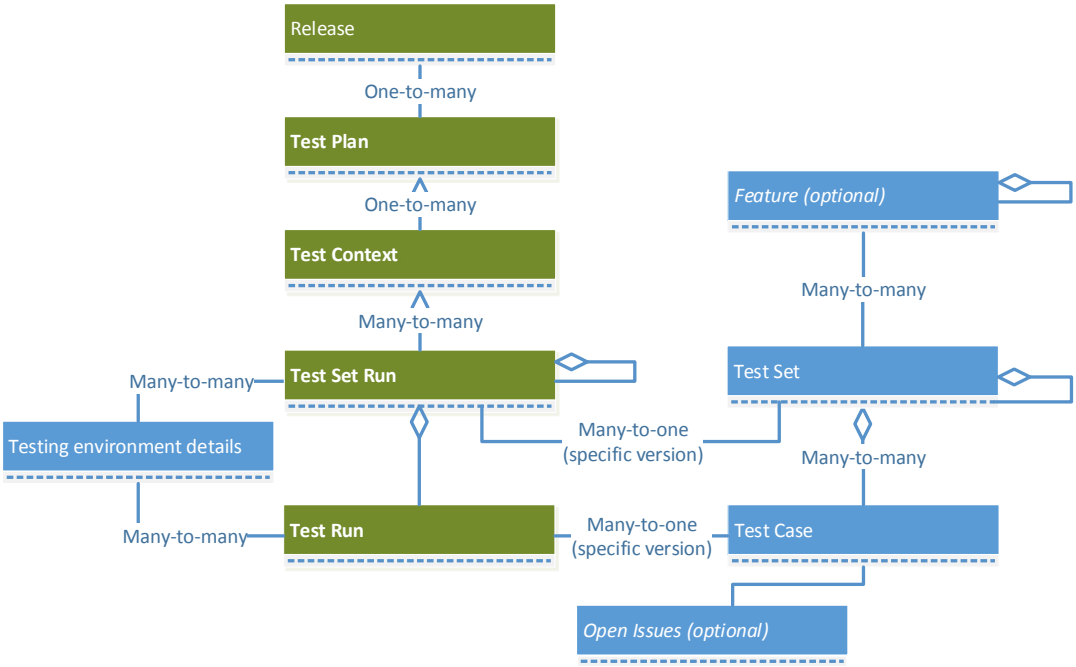


Figure 4.3 *An example structure of the test Management tool objects. The blue objects represent test cases and test suites, out of which the test run objects are created. Adapted from (M-Files Corporation 2015b).*

they add an approximation of the testing time to the test set metadata. These approximations are then used when planning and scheduling the system testing. Testing progress can be monitored from the metadata of a test set run. The test management tool automatically calculates the completion percentage to the test set run metadata based on the state of the test runs that are part of the test set run. The completion rate can be aggregated up until to the test plan object level so that the completion percentage is calculated from all test runs under that test object level.

4.3 Current testing metrics and problems in reporting

Currently, the testing related metrics that the organisation uses to evaluate the product development process are the number of errors found, the number of errors fixed, and the number of improvements completed. Lazic & Mastorakis (2008) calls these type of metrics "problem report information" metrics. These metrics are used in the planned milestone criteria that need to be fulfilled to ensure the quality of M-Files software release. The release date may be adjusted if the milestone criteria have not being met in the planned schedule. The defect metrics data is collected from defects' metadata in Tracker with M-Files Reporting tool which is an M-Files feature for visualising data in an M-Files vault. The M-Files Reporting tool is described in more detail in Section 5.3. An example report that is used in evaluating the milestone criteria is presented in Figure 4.5 (M-Files

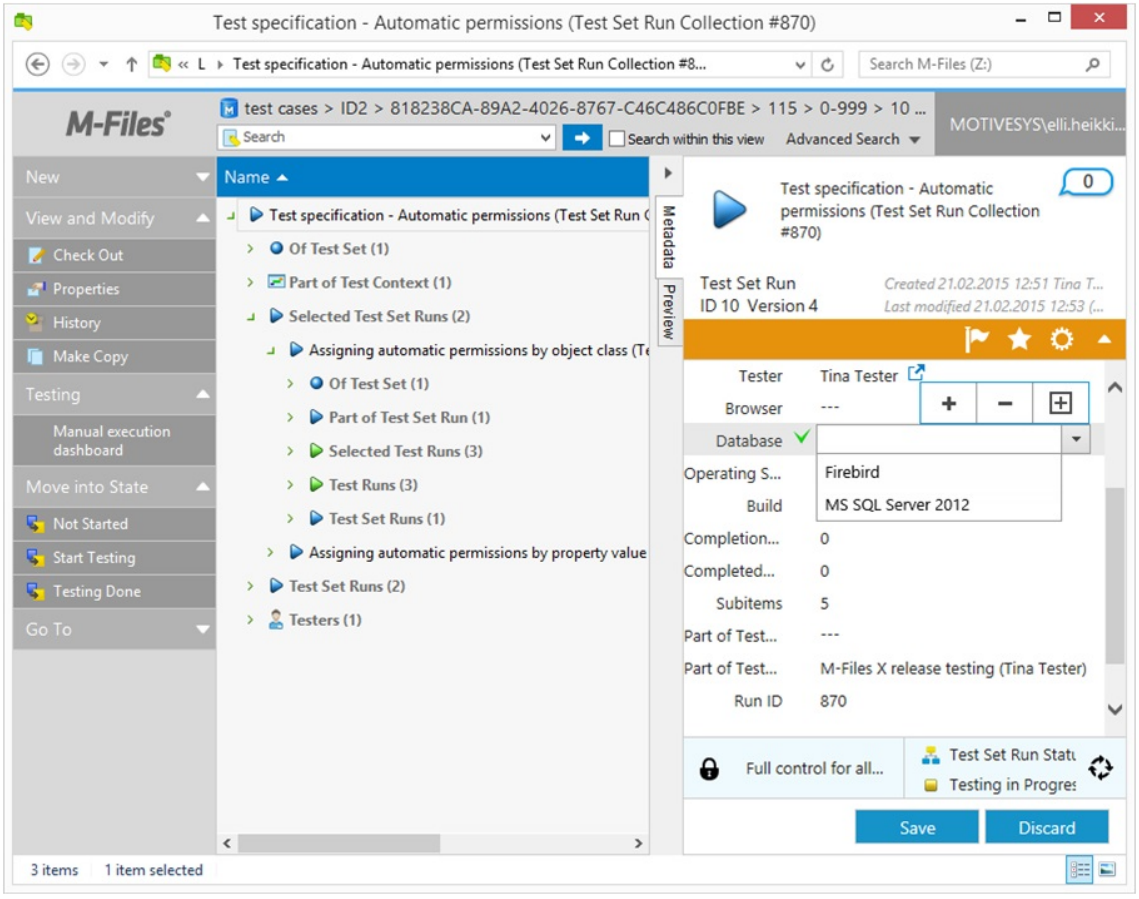


Figure 4.4 *Test run object hierarchy in Test management vault with the metadata card of a test set run collection showing.*

Corporation 2013)

During the implementation phase of the product development, monitoring the progress of user story testing and communicating the test results occur mostly via e-mail and live discussions. In the current test management approach, the testing team functions as a testing pool. In other words, testing tasks are prioritised and assigned to a next free tester, with certain heuristics that take into account the individual tester's experience with specific features. This is possible in the current organisation as the development and testing teams are mainly located in the same office and the number of testers is not too big for the test manager to manage. Therefore, no formal test plan or schedule are needed for user story testing. Currently, in general it is not required to report testing results of user story testing further than the R&D department management. Test results and any found defects are addressed within each user story as required by the Definition of Done (DoD). Therefore, besides the estimated testing time for each test set, no measurements are collected during the user story testing.

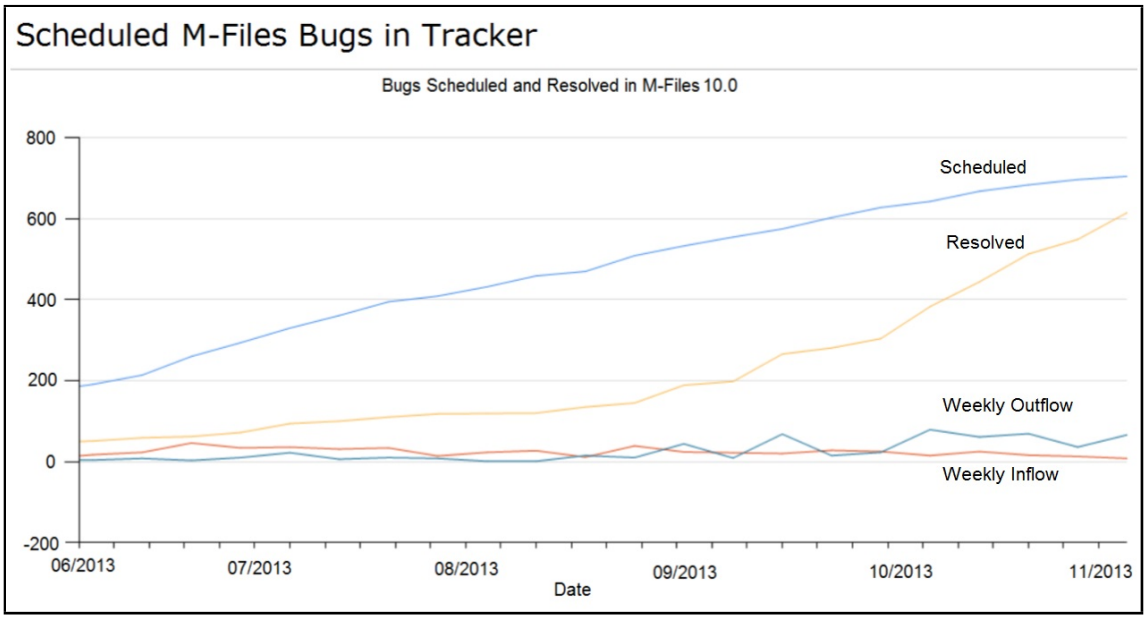


Figure 4.5 *A report of scheduled defects for release 10.0 in Tracker during system testing phase.*

In the system testing phase of the product development process, however, it is important to plan, schedule, monitor, and report the testing activities more carefully. Currently, the test plan for the system testing phase is a collection of Excel spreadsheets that include for example the information of what test sets are selected for the regression testing suite, assigned test environments, and the resources needed for testing. The test schedule document is also an Excel spreadsheet that includes the testers and test sets assigned for each day. The Excel sheet is then adjusted based on the actual progress of the testing and therefore offers limited visibility to the testing progress and when the testing will be done.

The practices in test management, test asset management and defect tracking are generally considered suitable for the current organisation. However, this is not the case with the current ways of managing the system testing phase. There is little tool support for efficiently communicating testing results and the methods are barely sufficient for even test planning and monitoring. It has also been noted that there is no effective way for various stakeholders, such as company management or auditors, to check the overall status of system testing of each release, namely the test run rate and pass rate. For the test manager and the company management, an effective communication of system testing progress and an approximation of when it will be done would be important. Currently, this information must be manually interpreted from the Excel sheets.

The need for better metrics for monitoring testing progress and evaluate testing effectiveness has been known in the organisation for some time already, but has not been fully addressed for various reasons. The main reason is that before taking the new test management tool into operation, the testing team used Word documents to save and manage

test cases. This made monitoring and reporting testing status very difficult as all data had to be collected from Word documents manually. The previous results of test cases were also very laborious to find in the document history. Therefore, metrics such as the number of executed test cases and the number of passed test cases were not collected very often. However, with the current system this data can be automatically gathered from the test case and test run objects' metadata with the help of the Reporting tool of M-Files software. The same approach is used in the company's defect management system. This would increase the transparency of testing effort greatly. There is also an increasing pressure to improve test reporting capabilities as the developed products and the organisation are quickly growing. Also, the recent development of M-Files QMS product variant has brought more and more customers from regulated field that have stricter requirements for test documentation from their vendor.

5. DEVELOPING TEST METRICS FOR TEST MANAGEMENT

The primary goal of this study was to develop a basic set of testing metrics to aid in test management, which is addressed in this chapter. First, the methods used to identify the desired testing metrics in the test management vault are described. Based on the found requirements, testing metrics are identified and implemented using the M-Files Reporting tool as it allows easy data extraction and visualisation in M-Files. The overall architecture of the feature and the process of creating metrics with it are described in Section 5.3. Finally, the created testing metrics are presented and evaluated in Section 5.4.

5.1 Semi-structured interviews and reviews

In this thesis, semi-structured interviews were used for identifying the requirements and wishes for the created testing metrics. The semi-structured interviews were chosen because despite having a prescribed list of themes or questions to be covered, they allow dynamic selection of questions varying from the interviewee to another and following the direction of discussion. Questions may be open ended and allow interviewees to use their own words to answer and additional questions can be asked in order to fully understand the answers. They are noted to be suitable especially for qualitative research, such as this study, and when there is a need to find out the reasons for interviewees' answers. They also allow learning something new from the interviewees and therefore ensure that all views are taken into account as the interviewer might not be able to capture everything in the questions. (Saunders et al. 2009, pp. 320-324) As the stakeholders interested in testing metrics consist of people with various job descriptions, it would not be reasonable to ask the same questions or take the same restricted point of view with all of them.

The selected themes for the semi-structured interviews were the following:

1. The interviewee's association with testing.
2. Requirements and wishes for testing metrics.
3. The purpose of the wished metrics and their priority.

These themes were selected while keeping in mind the IEEE process (IEEE 1998) standard discussed in Section 3.1 for developing metrics into an organisation, emphasizing the importance of identifying the objectives of metrics. The first theme covers the background information that helps to understand the requirements that different stakeholders may have for testing metrics, and how it depends on their job description. In addition to identifying the required testing metrics, one objective was for both the interviewer and the interviewee to understand more clearly for what purpose would the stakeholders need the metric and how they would interpret the measurements. As noted in Section 2.3.3, one of the main dangers with metrics is measuring the wrong things or measuring things wrong, the purpose of the last theme was also to draw the interviewees' attention to what they fundamentally want to know about testing, and if the metrics they proposed would serve that need. Furthermore, the objective was also to provide insight into how the presentation of the metrics should be developed so that the interpretation could be easy and accurate. The purpose is to take into consideration the challenges in metrics presentation which is also one of the themes in the IEEE process (IEEE 1998). The nature of semi-structured interviews allow to cover these topics in a dynamic order, and to build the answers during the interview.

The interviewed stakeholders were selected among the stakeholders within the organisation that are closely affected by testing and based on their experience in testing. As the goal of this study was to create only the most critical metrics for test management, the interviewees were mostly, with the exception of one, chosen within the Research and Development. This is because the created metrics were intended mainly for the needs of R&D personnel. Still, investigating the future development ideas was also highlighted. The list of interviewees is included in Appendix A. In the following sections, each interview is referenced using identifiers specified in Appendix A.

As the author of this thesis had been working as a tester in the organisation for over a year, the experiences and observations provided suggestions and ideas for metrics and their representation. Later on, during the implementation, the metrics were further reviewed in less formal discussions, also with people who were not initially interviewed such as other testers and developers. These discussions were used to validate the assumptions made by the author. They helped to confirm implementation details that were not yet known during the interviews, provided assurance that the correct metrics were being developed, and revealed possible defects in the implementation. All in all, the requirement collection and metrics development were highly iterative and largely based on regular reviews and discussions after the initial interviews.

5.2 Identified requirements for metrics

The data from the interviews were analysed to recognise the fundamental objectives for metrics and put them in the order of priority based on their importance and pre-evaluated level of difficulty. Prioritisation was done because the allowed time was limited and because it was acknowledged that not all identified metrics could be implemented during this project. Higher importance was given to metrics requested by the senior director of R&D and the test team manager. Higher priority was also given to metrics with a lower level of difficulty. The level of difficulty was lower if the data for the created metrics was already available, if it could be easily added to the test management tool of the organisation without big changes to processes or tools. All requirements were still collected for future references and to provide insight into where the processes and documentation practices should be developed into. The identified requirements for metrics, their objectives, and the interviewee(s) who requested the metrics are presented in Appendix B in the order of their priority.

As a summary, the interview results can be classified in two groups: either the interviewees specifically requested a certain metric or they had a more abstract objective that they had thought could be addressed with metrics. With the specific metrics requests, only some details regarding visualisation and granularity had to be discussed. Otherwise, they were ready for implementation as such. With the more abstract ideas, the root objective was investigated and potential metrics were planned. In some cases, it was concluded that the metrics can not be implemented with the designated tool or with the currently available information.

Because at the moment, test cases are almost the only unit of measurement available in the organisations test management tool, most identified metrics are based on the number of test cases. This is controversial to what the current opinion of test case metrics is. Metrics that are based on the number of test cases are highly criticised, because test cases are not equal in size, quality, or type (Craig & Jaskiel 2002, p. 85; p. 258). Such metrics may lead the attention only to the number, that does not directly tell anything about the quality of the software. However, having some metric is better than having none. As long as test case metrics are used knowing their flaws and checking also the quality of test cases, there should be no harm in them. Also, if some stakeholders or the traditions in the field require such metrics, there might not be other options. However, effort should be put into developing better metrics not settle with test case metrics.

Most of the specifically defined metrics were requested by the senior director of R&D or the test team manager. These included test pass and run rate in a timeline (I1), the number of test cases in a timeline (I2), the relative amount of testing in different test environments

(I2), the remaining test time of selected test set, and the number of test cases marked done each week (I3). The objective or usage for the test pass and run rate was to allow viewing the status of system testing in terms of observed quality and progress. These metric would allow seeing how the rates have been progressing in a certain time frame and estimate when the selected test suite would be done at the current pace. The metrics would be used mostly during system testing for monitoring the testing effort and managerial decisions. It may also be that the data is asked in audits. It was also noted that comparing the system testing of different releases might be interesting. (I1)

The number of test cases was requested so that it would be possible to know how many test cases there are currently, and how many have been created between releases. The test manager also wanted the number of test cases grouped by their state (Draft, Active, or Discarded). This was because test cases created by the subcontractors are first created in Draft state and reviewed before moving them to Active state to ensure their adequate quality. The reviews are done periodically when needed and the metric would help to estimate when the review should be done and how much effort would be needed with the current amount. Another metric requested by the test manager was a chart to show how much testing has been done or planned in each test environments, namely web browsers and operating systems. Its purpose would be to help planning testing in different environments to match their prevalence. (I2)

The objective of the number of test cases run each week was not as clearly defined and suggested partially out of interest. The suggested purposes included helping to get insight into the weekly throughput of testing and testing pace which would in turn help test planning. The problem is that it is difficult to include the number of testers that contributed each week. (I3) The objectives and the benefits with this metric were rather vague but as the level of difficulty was rather low, it was prioritised for implementation.

One requirement that was quite specific by nature was the time since a test set was last run. The root objective was to make sure there are no test sets that are not run for several years. This risk originates from the growth of the developed software and prolonged release cycle. After the size of the developed system has grown, the number of test sets has grown as well. Therefore, there have not been enough resources to include all test sets in each system testing phase. For many releases, the time between two major releases has been a year. However, the release cycle has been prolonged so that the current plan is to release a new version after two years. For these reasons, the test manager wanted a way to identify the test sets that have not been run for a long period of time so that the need to run them could be evaluated. (I2) However, the metrics was not implemented during this study, as the implementation format needed further specification and the time run out.

The most important requirement for testing metrics that was more abstract, was developing metrics that would help test planning in terms of scheduling, monitoring, and prioritising (I2). This need has been recognised in the organisation before and part of the problem had been the lack of a test management tool (Pohjoisvirta 2013, pp. 62-64). At the time of this study, a new test management tool that can make the metrics improvements possible had been developed and taken into use. However, during this study, the tool had been in use for less than a year and both the tool and the processes were still subject to further development. The three main goals were to (1) have better ways to track testing time, (2) have a more formal way to prioritise features for system testing, and (3) have a better tool for planning and scheduling testing. Out of the three, the second goal was most relevant in terms of metrics. The others were mostly about tool support and creating required information.

It was noted that currently, testing priority of features was decided by discussing the matter with various stakeholders of each features importance and experienced quality during user story testing. It was discussed that the observed quality during user story testing could be used for prioritising. Example measures include the amount of defects found, testing time, the size of the feature for example in the number of user stories or story points, and test pass rate. With already released features, testing priority could also be evaluated based on the importance of a feature to customers, for example with surveys to customers, to sales personnel, and consultants, or from defects reported by customers. However, it was concluded that it would require either information that is not currently available, or combining information from outside the test management tool, such as ScrumWorks Pro (CollabNet 2015), the Agile planning tool used in product development management, and Tracker, the defect management vault in M-Files. With current practices and existing information it would not have been possible to couple the information from different locations with reasonable amount of work during this study.

There were also other requirements discussed in the interviews but evaluated as out of the scope of this study. These include pairwise testing and developing metrics to support that (I3) and a risk model and creating metrics to support the model (I4). In this thesis, they were prioritised low as the current testing practices do not include pairwise testing, and developing a risk model was not in the scope of this thesis. Also, in the current organisation, many factors that could be used in the risk model cannot be measured due to insufficient process and tool support. Therefore, they were left as future development ideas, but these improvements should be kept in mind even when creating the basic metrics.

5.3 Implementing metrics with M-Files Reporting

Metrics were developed using the M-Files Reporting feature that includes automatic metrics data collection and visualisation as reports. It should be clarified that even if the tool is called 'Reporting tool' and the data visualisations are called 'reports', they are essentially a way to implement metrics or process numeric data and not what a term 'report' generally might suggest. An example of a report implemented with the M-Files Reporting feature is shown in Figure 5.1. The tool was a natural choice for several reasons. It allows automatic and simple data extraction directly from the M-Files based Test Management tool. It supports easy-to-use tools for performing the data analysis and visualisations and finally, allows viewing the metrics directly in M-Files clients upon request. Also, because the tool has been developed internally, it was easy to find documentation and technical support for implementation.

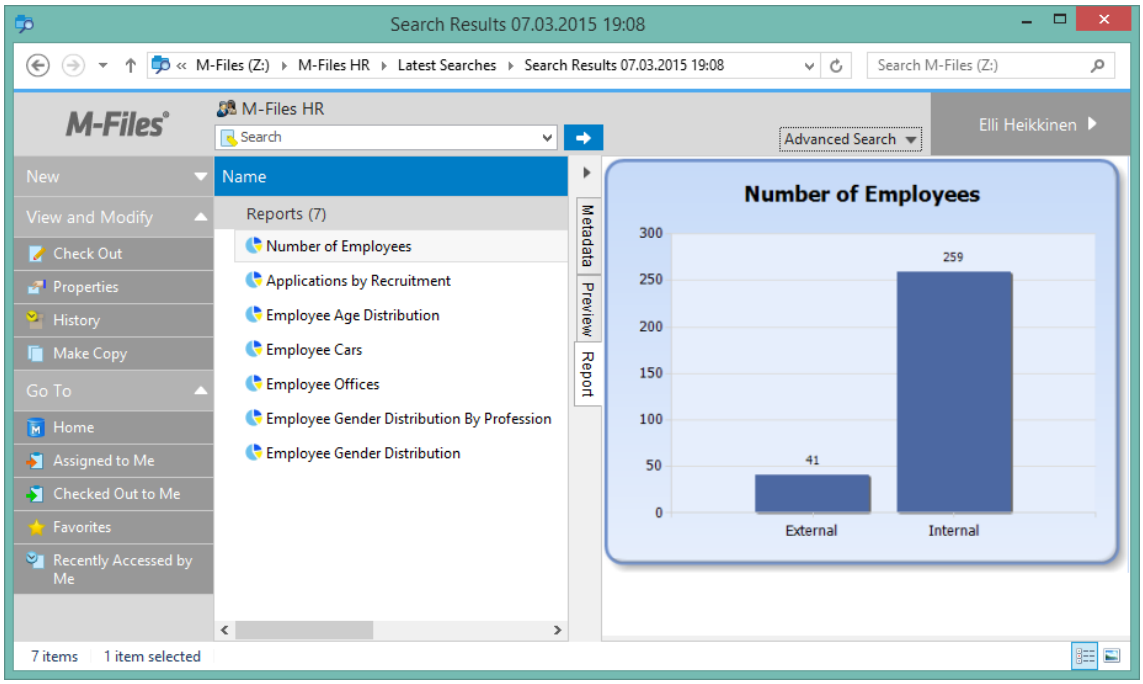


Figure 5.1 *A report object in M-Files Desktop is selected and the report shown in the right panel. The report visualises the number of employees in the organisation currently, classified to internal and external employees.*

In a nutshell, M-Files Reporting feature exports selected data to an external Report Data database. From there, the data can be queried with SQL and for example statistically analysed, visualised, and displayed in M-Files client or in third-party applications. The high level architecture of the M-Files Reporting tool is illustrated in Figure 5.2. An administrator can specify the extracted data and run the extraction process either manually or according to a predefined schedule. M-Files Reporting Data Services exports the selected data from the M-Files vault database to an external Report Data SQL database. Then, the

data analysis and visualisations can be designed with report development tools, namely Microsoft's Report Builder (Microsoft Developer Network 2008b), which is free of charge and allows small or medium-sized reporting projects, or Business Intelligence Development Studio (Microsoft Developer Network 2008a), which supports the creation of more complex reports better. In this project, Report Builder was used due to its simplicity. The developed reports are then uploaded to the Report Storage database and on request, rendered and displayed by the Microsoft SQL Server Reporting Services (SSRS) infrastructure (Microsoft Developer Network 2008c). The reporting service connects to the Report Data database to which the data from M-Files were extracted and renders the report when it is requested by the end user via M-Files client. (M-Files Corporation 2014a)

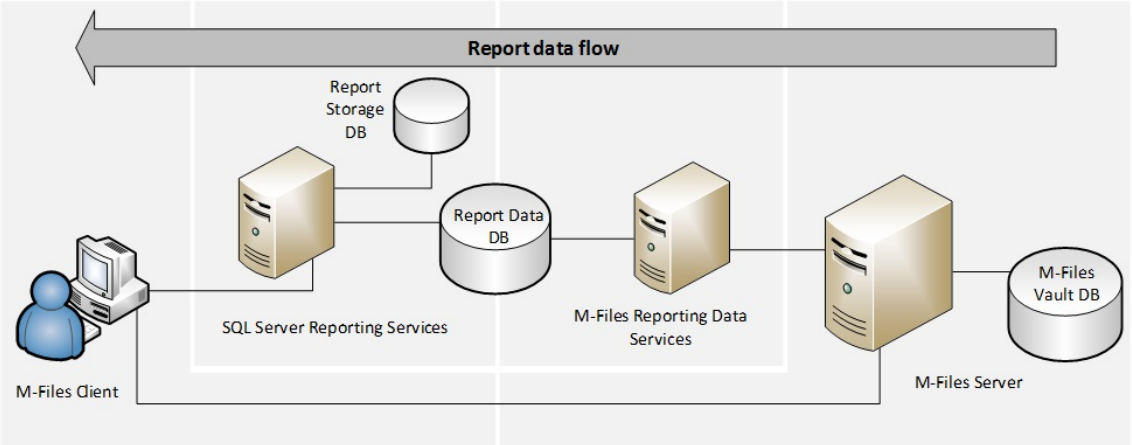


Figure 5.2 *M-Files Reporting tool structure and data flow (M-Files Corporation 2014a).*

The creation process of a report with the reporting tool goes as follows:

1. Selecting and exporting data from M-Files to Report Data database

The data set required for a report is selected and exported to an SQL server using a Data Set Export user interface in M-Files Admin tool. Users select the needed objects and their properties to be exported. It is possible to apply some filters, or export the change history. The export can be scheduled to occur at certain intervals. There can be several data set export jobs but each data set must be exported to a separate database. An example of data set selected for export is shown in Figure 5.3.

2. Querying exported data from Report data database to Report Builder

In this study, reports were created using the Report Builder. After the data has been successfully exported from M-Files to the Report Data database, the data can be fetched from the database to Report Builder using SQL queries. Depending on the nature of the data and reports, one or more data sets are needed for one report. For most of the reports created during this thesis, a possibility for selecting the displayed groups of data was

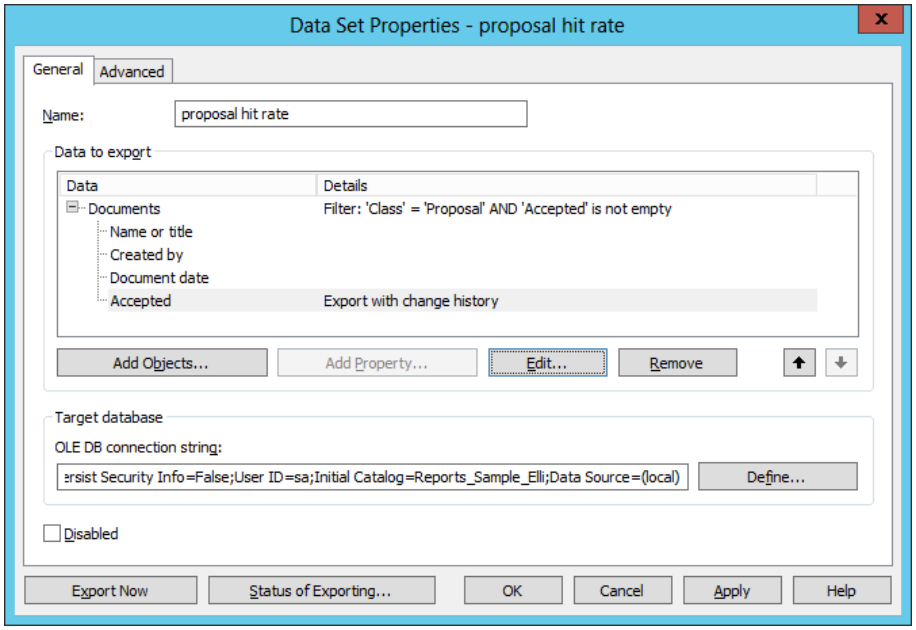


Figure 5.3 *M-Files Data Set Export selection dialog. The data selected for export consists of the selected properties of Document objects that are of Class 'Proposal', whose 'Accepted' property is not empty.*

wanted. For the selections, separate datasets were required. For performance reasons, it is better to do as much of the data processing, such as summing and counting with SQL queries as possible. This is often the computationally heaviest part of a report rendition. Luckily, modern database tools make designing the queries easy. Report Builder provides a graphical query designer for developing the most basic queries for users that are not familiar with SQL.

3. Designing the report with Report Builder

After creating the datasets, the reports can be designed wither with Report Builder or Business Intelligence Development Studio. In this thesis, Report Builder was used. Report Builder's user interface for designing reports is shown in Figure 5.4. Report development includes calculating presented data values from raw data with expressions written in Microsoft Visual Basic, adjusting the displayed data set with parameters and filters, and fine tuning report labels, legends and other appearances. Report Builder supports various different charts and tables for visualising data and offers numerous useful features for reports, such as filtering the data based on the selected objects in M-Files, links to the actual objects displayed in reports, and more.

4. Publishing the report and viewing via M-Files Desktop

The created reports are then uploaded to the SQL Server Reporting Services using its web interface. From there, the reports are accessible with an URL. To view the reports

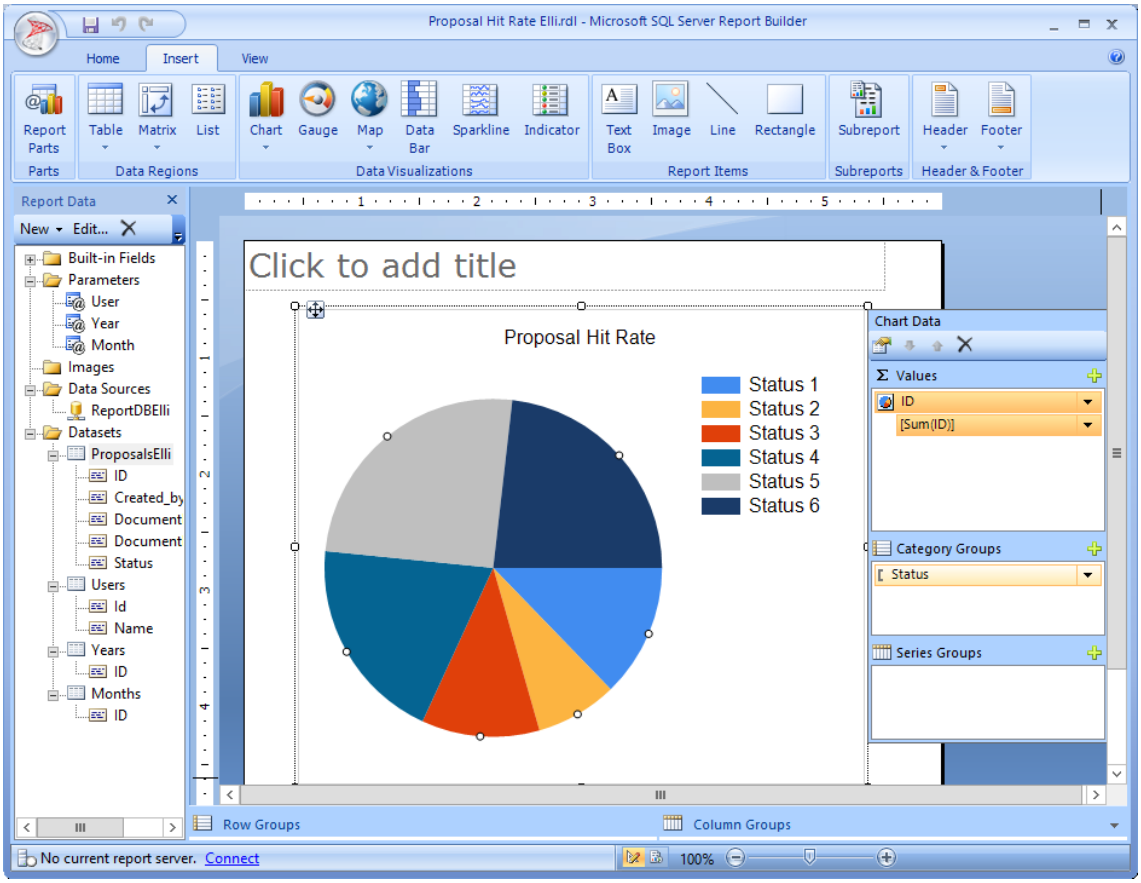


Figure 5.4 *Report Builder's report designing view.*

in M-Files Desktop, the user needs to create an object of type Report, that is a built-in object type in M-Files. The URL where the report is available is set in the properties of the reporting server. The report is then shown whenever the report object is selected as in Figure 5.1.

This way, the metrics data collection, data analysis, and visualisation can be entirely automated and are available all the time.

5.4 Developed metrics and their evaluation

In total, seven metrics and their presentation as reports were implemented during this study. The identified metrics requirements, the developed reports, future development ideas, and related stakeholders are summarised in a metrics data sheet in Table 5.1. Having a metrics data sheet is part of good practices as described in Section 3.1.3. It helps in maintaining the metrics set as it allows to keep track of the existing metrics, understanding of each metrics purpose, and the future development ideas. From the data sheet, it is easy to periodically evaluate each metrics value. Next, a short description of each developed report is given and the ability of the report to meet the requirements is discussed.

Table 5.1 *Metrics datasheet*

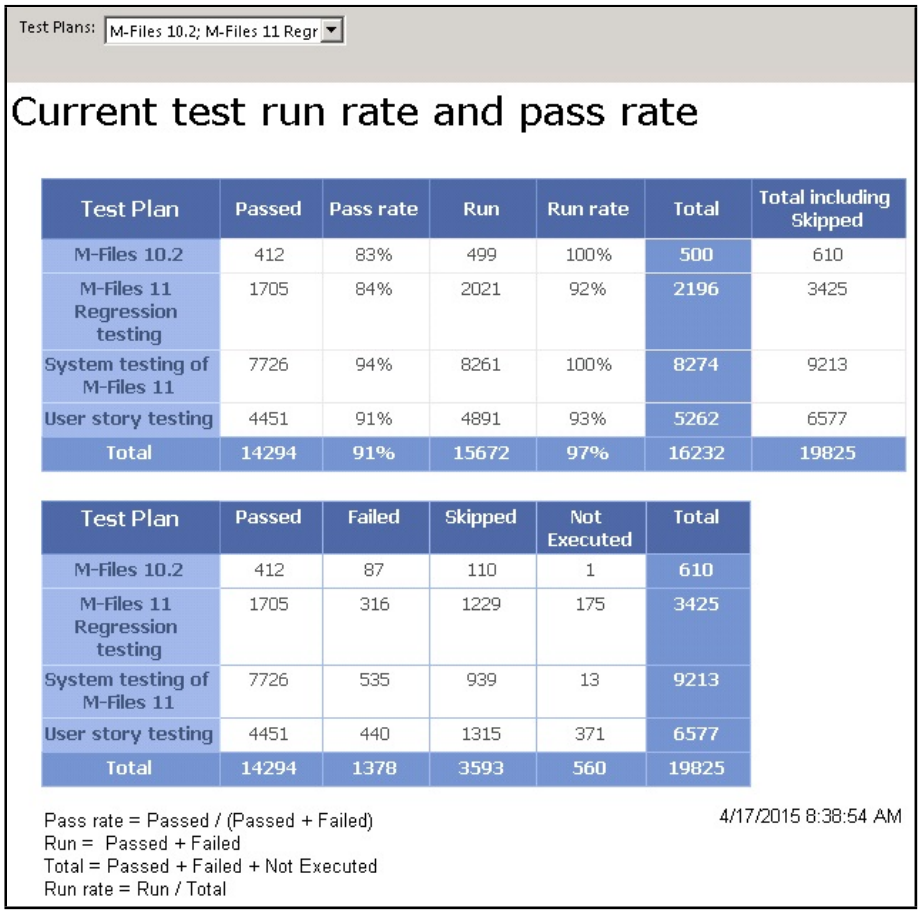
Purpose	Current metrics	Metrics ideas	Stakeholders
Test scheduling	R3: The number of test cases by state in a timeline R6: Remaining testing time by test plan	-	Test manager, Testing team
Test environment planning	R4: Usage of web browsers in selected test plans R5: Usage of operating systems in selected test plans	-	Test manager, Testing team
Test prioritising	-	Feature Risk: - Number of found defects - Size - Novelty	Test Manager, Senior Director of R&D
Test monitoring	R6: Remaining testing time R2: Test run and pass rate in a timeline R7: The number of test cases executed each week	-	Test manager, Senior Director of R&D, Testing team
Test reporting	R1: The current test run and pass rate by test plan	-	Senior Director of R&D
Test case management	R3: The number of test cases by state in a timeline	Test automation status	Test manager

The current run rate and pass rate by test plan

In this report, see Figure 5.5, the table above includes the number of passed and run test cases, the pass rate and run rate, and the total number of test cases. The total is specified both not including the skipped test runs (Total) and including the skipped test runs (Total including Skipped). The table below contains a breakdown of test runs in each state (Passed, Failed, Skipped, Not Executed) by test plan. The tables represent the current situation. The shown test plans can be selected with a menu above the report. Below the tables, the note explains how the fields are computed. This report is for quick access when one is interested in the latest test run and pass rate only or wants to compare results from different releases.

Run rate and pass rate trend

The second report, see Figure 5.6, visualises the weekly pass rate and run rate of the selected test plan in a line chart. It also has a more thorough breakdown of the testing status in a table under the chart including for example the number of tests passed, failed, and the total number of test runs. The visualised test plan and the time frame can be selected with the menus above the chart. Because it shows the weekly values and not only



Test Plans: M-Files 10.2; M-Files 11 Regr

Current test run rate and pass rate

Test Plan	Passed	Pass rate	Run	Run rate	Total	Total including Skipped
M-Files 10.2	412	83%	499	100%	500	610
M-Files 11 Regression testing	1705	84%	2021	92%	2196	3425
System testing of M-Files 11	7726	94%	8261	100%	8274	9213
User story testing	4451	91%	4891	93%	5262	6577
Total	14294	91%	15672	97%	16232	19825

Test Plan	Passed	Failed	Skipped	Not Executed	Total
M-Files 10.2	412	87	110	1	610
M-Files 11 Regression testing	1705	316	1229	175	3425
System testing of M-Files 11	7726	535	939	13	9213
User story testing	4451	440	1315	371	6577
Total	14294	1378	3593	560	19825

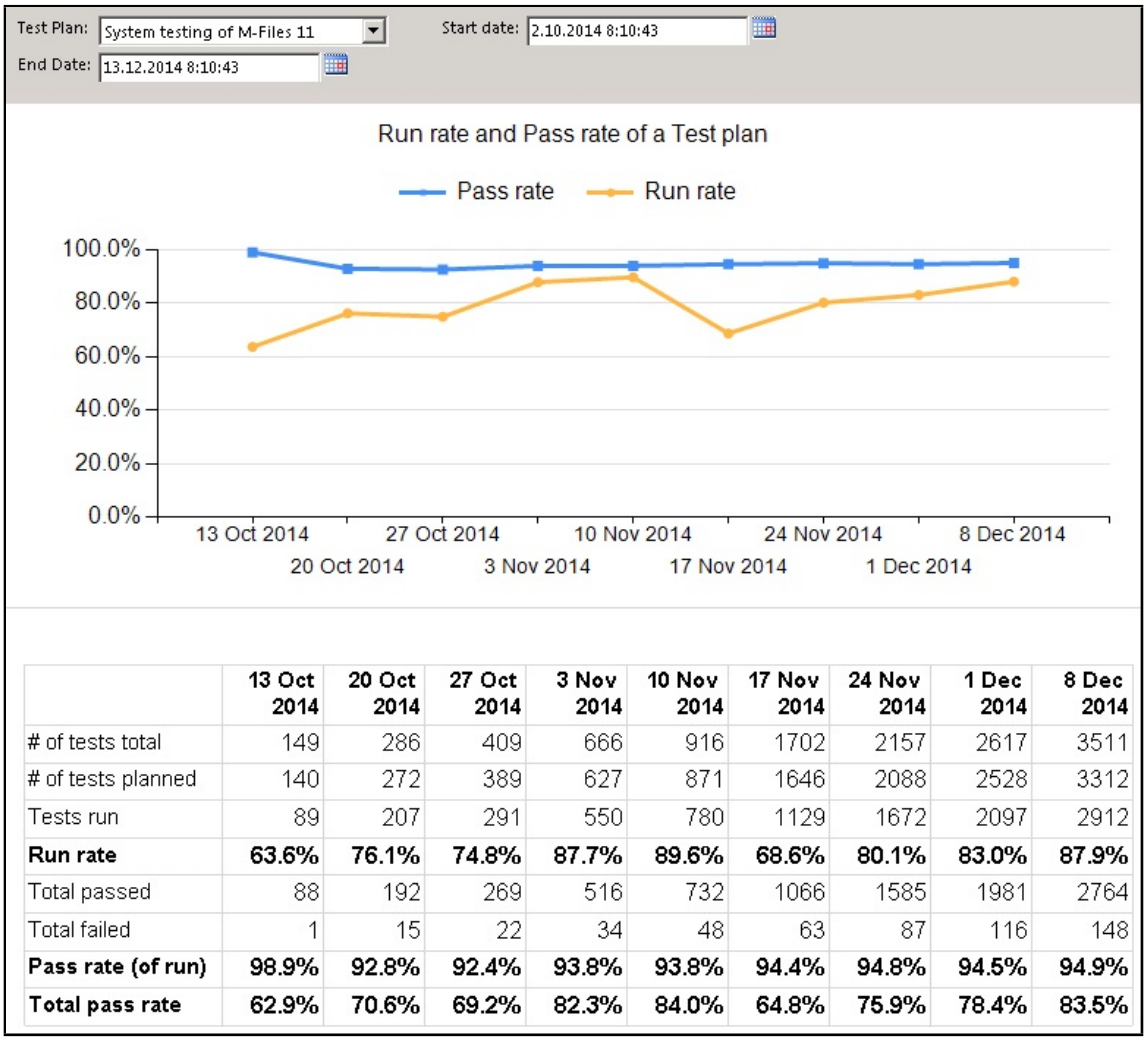
Pass rate = Passed / (Passed + Failed)
Run = Passed + Failed
Total = Passed + Failed + Not Executed
Run rate = Run / Total

4/17/2015 8:38:54 AM

Figure 5.5 *The table above is the current test run and pass rates by test plan. The table below, is a breakdown of test runs currently in each state by test plan. The tables can be filtered by selecting the shown test plans.*

the current values, this report allows viewing only one test plan at the time. It may be argued that the first two reports are complementary, or that the first report does not offer any incremental value to the second. However, as noted, the first report allows seeing the current situation quickly and easily. It also allows an easy way to compare the final situation of different test plans, for example the result of different releases.

A drawback of this report is that currently, all test set runs are not created right away at the beginning of testing. Instead, the test runs are added as the previous test sets get done. The root reasons for this may be in the lacking tool support for easier test planning and adjusting the plan. It follows that the run rate goes up and down when new test runs are added and the rates do not actually depict the full situation correctly. In a way, the chart meets the requirements for monitoring test status with a quick glance, but it does not provide insight into when the testing will be done at the current pace. This is a known issue and it was left open whether the report or the current practices need adjustment.



	13 Oct 2014	20 Oct 2014	27 Oct 2014	3 Nov 2014	10 Nov 2014	17 Nov 2014	24 Nov 2014	1 Dec 2014	8 Dec 2014
# of tests total	149	286	409	666	916	1702	2157	2617	3511
# of tests planned	140	272	389	627	871	1646	2088	2528	3312
Tests run	89	207	291	550	780	1129	1672	2097	2912
Run rate	**63.6%**	**76.1%**	**74.8%**	**87.7%**	**89.6%**	**68.6%**	**80.1%**	**83.0%**	**87.9%**
Total passed	88	192	269	516	732	1066	1585	1981	2764
Total failed	1	15	22	34	48	63	87	116	148
Pass rate (of run)	**98.9%**	**92.8%**	**92.4%**	**93.8%**	**93.8%**	**94.4%**	**94.8%**	**94.5%**	**94.9%**
Total pass rate	**62.9%**	**70.6%**	**69.2%**	**82.3%**	**84.0%**	**64.8%**	**75.9%**	**78.4%**	**83.5%**

Figure 5.6 *Above, a line chart of weekly test run and pass rates of selected test plan. Below, a weekly breakdown of the same data in more detail. The visualized test plan and the time frame can be selected above the chart.*

The number of test cases by state

This report, see Figure 5.7, is a stacked area chart depicting the trend of the number of test cases by their state. The visualised time frame can be selected using the menus above the chart. There is a limited amount of benefit this report can offer but it does answer to the questions of how many test cases there are currently, how many were created during a certain time frame, and how many test cases are in Draft state waiting for a review. It has been noted that the organisations test assets need a sweep to discard duplicates and discard or update outdated test cases (Pohjoisvirta 2013, p. 60). This report may offer interesting information of following the progress and results of such sweep if it were to happen, for example by illustrating the amount of test cases being removed or moving to other states.

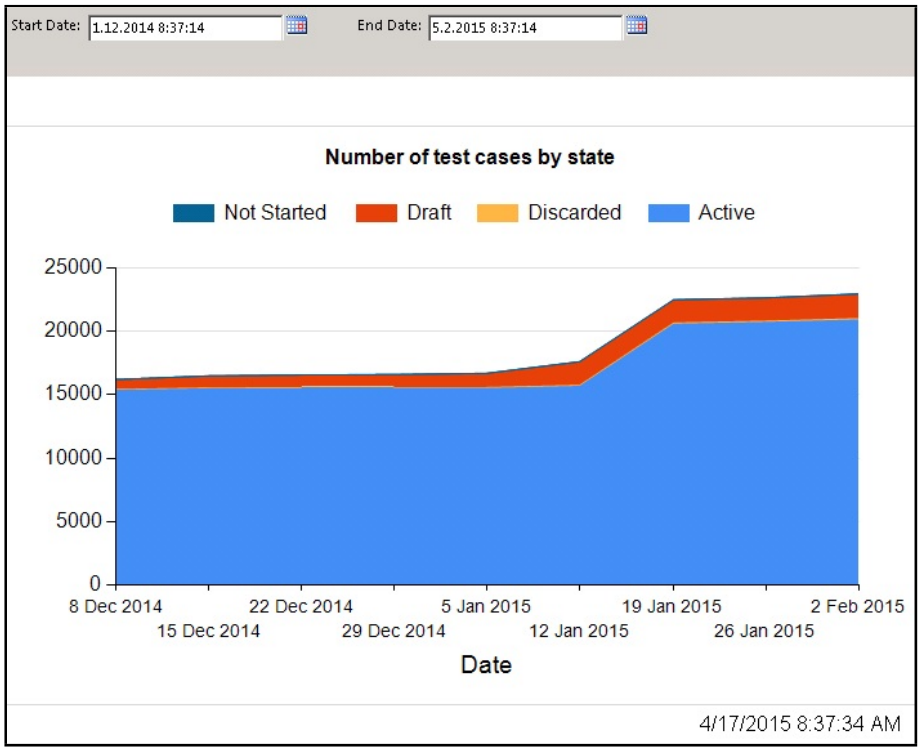


Figure 5.7 *The number of test cases weekly by grouped by their state. The visualized time frame can be selected above the chart.*

It was noted during the reviews that this report needs more information to be useful. The test cases should be classified also in other ways than by their state, for example the automation status, or by the feature they test. It could be useful for estimating the required manual testing effort, how it differs from the last release, or the amount of both manual and automated testing overall by feature.

The amount of testing with different web browsers in selected test plans

This report, see Figure 5.8, illustrates the amount of testing in different web browsers in the selected test plans in a pie chart. The section size represents the portion of test runs within the selected test plans that have the specific web browser as their test environment. The chart does not take into account whether the test runs have been run or not. Therefore, it allows seeing what the proportional coverage of each browser will be with the current plan, regardless of the testing progress. From the chart, it is easy to see which browser should be used less and which browser should be used more. Specific browser versions are specified in the test runs, but different browser versions, except Internet Explorer (IE) versions, are grouped together in the report. This is because there are customers who use a specific IE version, for example IE 8, but other browsers are usually always updated to their latest version. Therefore it is necessary to see the test coverage of each browser in general and the specific IE versions.

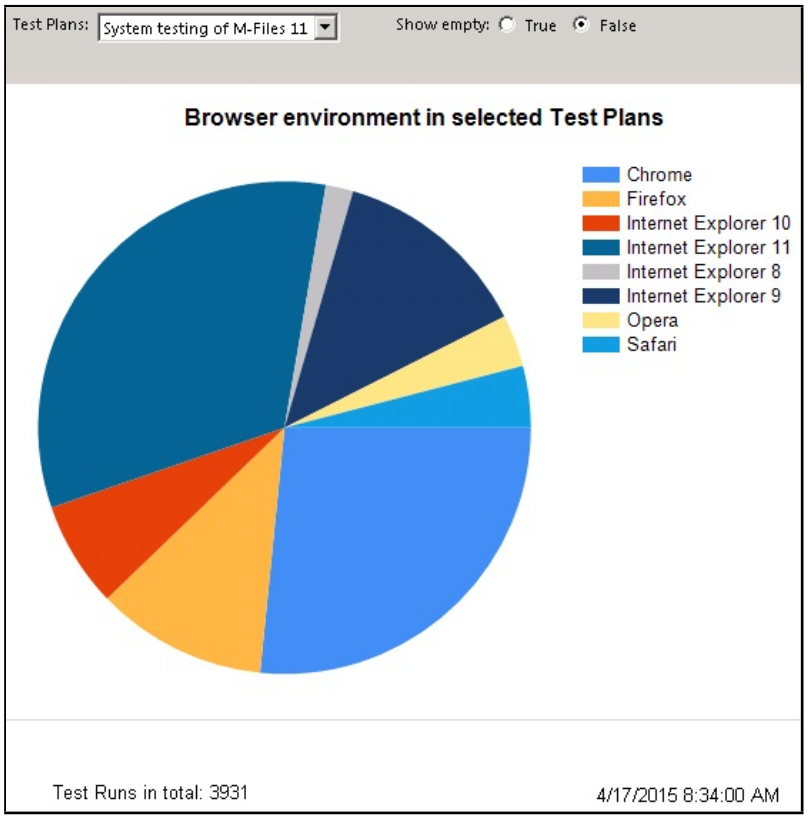


Figure 5.8 *The usage of different Web browsers in selected test plans. Other than Internet Explorer versions are shown as a single value. The included test plans can be selected above the chart. Additionally, the radiobutton allows selecting whether to show test runs where the browser property was empty.*

The dropdown menu above the chart allows selecting the shown test plans. The usual case is probably to select only one test plan, for example 'System testing of M-Files 11'. However, one may want to see the overall coverage of different test environments. The radiobutton on the right allows selecting whether to show the portion of test runs with browser environment undefined or not. The case may be that the test environment is yet to be decided for those test runs, and showing the amount of test cases still without environment would allow checking the amount of them. On the other hand, a more usual case is that the test plan consists of test runs in both M-Files Desktop and M-Files Web clients. As the browser property is not applicable in a Desktop environment, the current practice in the organisation is to leave the browser property empty. Therefore, test runs without a browser environment include the test runs executed in M-Files Desktop client. Assuming that most web client test runs are assigned a web browser, the amount of test runs with an empty browser property would be the portion of M-Files Desktop testing in the test plan. This was not addressed in the initial interviews but the option was left open for further specification. The right approach could be to explicitly specify a 'Not applicable' value for browser in M-Files Desktop test run so that the actual amount of test runs with web browser environment not yet defined could be recognised.

The amount of testing in different Operating systems in selected test plans

This report, see Figure 5.9, is a similar pie chart as the previous one and illustrates the relative usage of different operating systems as test environment in the selected test plans. As in the previous browser coverage chart, this chart shows all test runs belonging to the selected test plans grouped by their selected operating system environment, also regardless of whether the tests have already been executed or not. Its purpose is the same as with the web browser chart: to help assigning the test environment to meet the targeted coverages. In contrast to the web browser usage chart, this chart includes the 'Empty' values by default. This is because regardless of the tested M-Files client, operating system environment is always applicable.

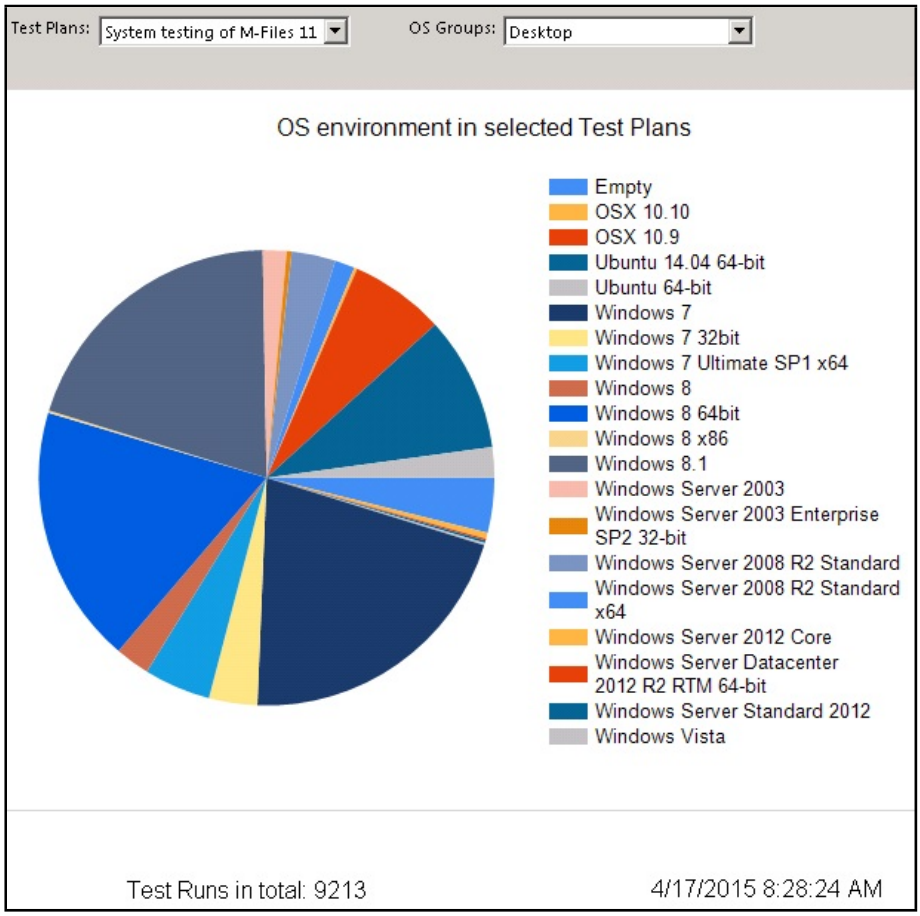


Figure 5.9 *The usage of different operating systems in selected test plans. The included test plans can be selected above the chart. The OS Groups selection allows selecting whether to show Desktop operating systems or mobile operating systems.*

The dropdown menu on the left above the chart allows choosing the included test plans, similarly as in the previous report. The other dropdown menu on the right allows choosing whether to include only the Desktop operating systems or the Mobile operating systems (such as iOS 7), or all. The usual use case would be viewing only Desktop operating systems, as there is not much to be done with the mobile operating system information.

In other words, the mobile test environment is planned based on the done user stories and releases, not the previous test environments. However, the option was left open as it offers some nice to know possibilities: the relative amount of testing in different mobile devices and when both Desktop and Mobile operating systems are included in the chart, it illustrates the relative amount of mobile testing versus desktop testing.

Currently, the value list for operating system property is not centrally managed. Instead, all users are allowed to create new values for operating systems list. As a result, the data in the test management vault is ambiguous and contains duplicates. It follows that there are many groups in the chart and it may be difficult to interpret. The chart also does not separate M-Files desktop client testing from M-Files web client testing. This is because currently, it is difficult to make the difference between the target clients as the environment information is scattered in various places, such as the test set name, the test context name, the feature, or the browser property. This is an issue that is known in the organisation and should be addressed at some point.

Also, it needs to be remembered that both the browser and operating system coverage charts have their limitations. Even if a certain test environment reaches a desired proportion in the test plan, it does not mean that enough testing has been done in that environment, or if correct test sets have been run in that environment. The total number of test runs included in the chart may strengthen the assumption but as the number of test runs itself is a questionable metric, great care should be taken when interpreting this chart.

Remaining testing time by test plan

This report, see Figure 5.10, is a table of test plans and their remaining testing time in hours, man-days, and working days. Man-days are computed by dividing the testing time by the length of a normal working day (7.5 hours). Working days are calculated as the number of man-days divided by the specified number of testers. This report is based on experienced testing time. When a test set is run, testers manually set the testing time in hours to the test set properties. The report calculates a sum of the estimated testing time values from the test sets, that are have not yet been completed, grouped by test plan. The purpose of this report is to help calculating the sum of testing time in a selected test sets. It can be used when for checking the remaining testing time of the selected test sets. This report was created mainly for the purpose of getting an approximation of the needed testing time automatically. This metric is arguably better than the number of test cases as it takes into account the 'size' of test cases and is based on an experienced testing time.

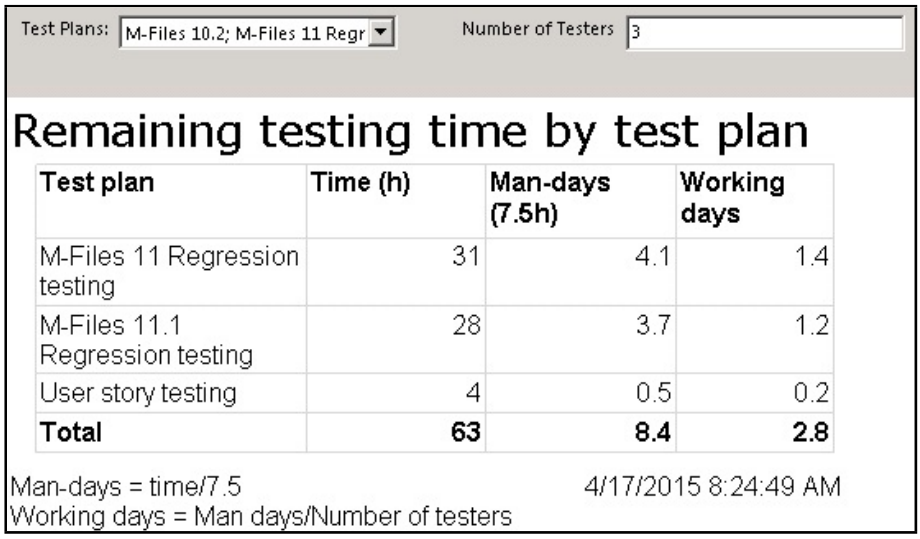


Figure 5.10 *The currently remaining testing time in hours, working days, and calendar time by test plan. The shown test plans and the number of testers can be selected above the chart.*

However, the metric also has some limitations. Firstly, not all test sets have a testing time set for them. This is a known issue and there are several reasons for it, one being the testers forgetting to add the time estimates after testing. Until the issue is addressed, it should be acknowledged that the time estimate is not accurate and is missing many values. One way to mitigate this problem could be to illustrate how many test sets or test runs in the test plan are without a time estimate. A second problem related to the lack of testing time estimates is the lack of reliability even if there was a time estimate. This is because there are no unified rules as to how to estimate testing time and each tester may use different principles to count the hours. The testing time depends on the tester. An experienced tester may be considerable faster than a newly recruited tester who is not yet familiar with the product. There is also no way to know how much efficient testing time testers have during their day as there are always interruptions and meetings. Due to these reasons, it is often recommended to use calendar time in testing time estimates (Kan 2002, p. 321). An automated approach to evaluating the testing time could help solving these problems, as discussed in the interviews. (I3)

Finally, as the testing time is set on a test set level, the remaining testing time is count based on the status of the whole test set run, not individual test runs. In other words, if there is even one test that is not run in a test set run, the test set run is not regarded as done, and the testing time of the whole test set is included in the time estimate. This may cause a notable error limit if a very big test set is in progress. While the test set run is in progress, the testing time does not decrease until the set is completed. One way to smooth out this issue could be to evaluate the remaining time based on the completion rate of the test set. However, it would bring back the problem of different test case sizes. Currently, this issue was deemed acceptable until a better approach is found.

The number of test cases executed each week

The last report, see Figure 5.11, illustrates the amount of test cases run each week by a test plan in a stacked bar chart. A test is considered as run during the week if it was moved to its current state during the week, and the state is Passed, Failed, or Skipped. Skipped test runs are included in this measure because they were included also in the total amount of test runs at the time of test planning. The test runs are grouped by the test plan they belong to and one can choose the shown test plans with the dropdown menu above the chart. The shown time frame can also be selected with the menus.

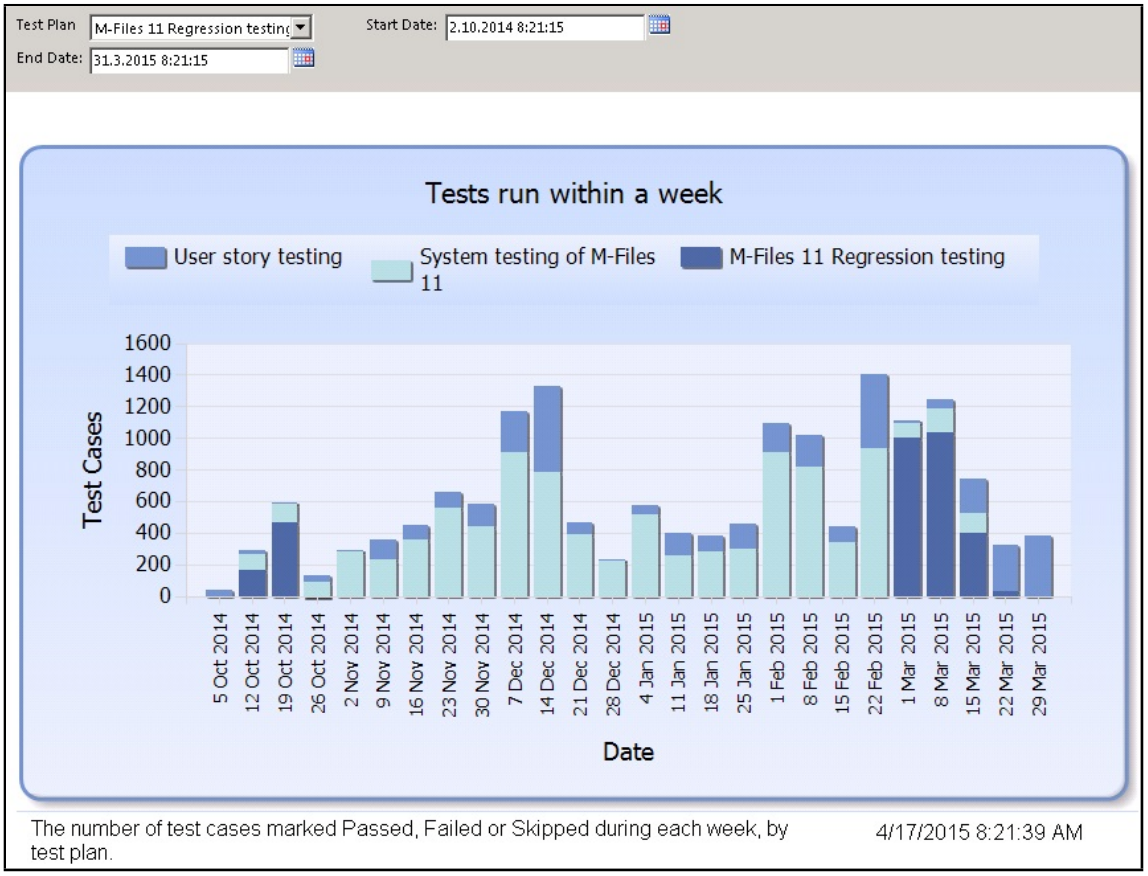


Figure 5.11 *The number of test cases marked done each week by test plan. The visualised test plans and the time frame can be selected above the chart.*

As noted in Section 5.2, this metrics was suggested more out of interest than a known objective. The purpose of this metric was to see what is the weekly throughput of testing which could help estimating the testing pace. However, the benefits may lose to the problems with this report. It uses the test case as a measurement unit, which is, as discussed, a highly criticised measure. Additionally, the report does not include information on how many working hours, in other words how many testers, were needed to achieve the results each week. On average, the report may offer some insight into how many test cases the testing team is able to run weekly, given that the team size stays relatively

constant. However, testing team is currently going through rapid growth. Also, as the report does not take into account who are executing the test cases, the results are affected by the subcontractors as well and their working hours are even more difficult to take into account.

6. EVALUATION

In this chapter, the results achieved during study are summarised and evaluated. Also, the quality and limitations of this study are discussed. Finally, some future development suggestions are described.

6.1 Evaluation of the results of this study

As a result of this study, a set of basic metrics for the most urgent needs were developed. The created metrics are by no means the most ground breaking or best metrics there are. Actually, they have quite limited abilities if they are to be used correctly. However, the developed metrics offer a good basis for developing better metrics. They may also help identifying other useful metrics that the stakeholders did not yet identify during this study, after they have seen what can be done. The capabilities and the quality of the developed metrics were analysed in more detail in Section 5.4.

In addition to the developed metrics, this study resulted in a lot of other useful information for the organisation such as metrics ideas for future development, and process and tool improvement ideas. The metrics ideas that were not possible to implement during this study offer good insight into which parts of the testing process and tools should be improved next and how, and what should be the goal in the long run. Analysing how the metrics could be developed helps to recognise important improvements that act as gatekeepers to higher level improvement. For example, it was noted many times that the lack of a centrally managed feature list prevents mapping the test cases, defects, and customer tickets to a functionality in a consistent manner. This is partially also due to lacking tool support. Whether that is good or bad, as many of the used tools are based on the company's own product, possible tool improvements are totally up to the company itself.

This study showed that there are numerous aspects the organisation can improve its testing and other quality assurance efforts. It was noticed that practically the only unit of measurement at the moment is a test case, even though it is a highly criticised measurement unit. Much of the required information for any other metrics was missing or even unsupported by the current tools and processes of the organisation. For example, testing time had not been set for most test sets, even though the test management tool supports it.

Therefore, it was noted that an automated system for monitoring testing time is needed. Another example was that currently, the test environment details are not necessary explicitly and consistently added to test runs but there were duplicate and ambiguous values, or no way to specify that some detail, for example a browser environment, was not applicable. Many small improvements that were identified as a by-product of this study were implemented into the test management tool during this study.

Through the literature review conducted in this study, the organisation gained knowledge of metrics. The organisation is now more familiar of how testing metrics and metrics in general should be developed, what should be taken into account when creating and using metrics, and what kind of dangers exist with metrics. The literature review revealed what kind of metrics other organisations use and how the practitioners currently view them. This information is useful when the organisation continues to improve the existing metrics and develop the metrics that were identified but not implemented during this study. Finally, the author is now much more familiar than before with the M-Files Reporting tool which makes the learning curve shorter when developing more metrics with it.

6.2 Quality and limitations of this study

Firstly, this research was essentially a case study, which means that the results are by no means applicable in other organisations. However, the methodologies, used tools, and the observations made during the study can be useful in similar projects as well as in the future development of metrics in the case organisation. This study was also conducted in a practical environment, in which there was a high pressure to produce concrete results from the sponsoring organisation. This thesis was produced in a relatively short time, which can be deduced from the evaluations of the developed metrics. Many were noted to still be in need for further development. The metrics development was highly subject to the limited availability and format of data in the organisation. While it was good that this observation highlighted the importance of certain improvements to the organisation, it considerably limited the possibilities of developed metrics. All in all, this study could have benefitted from a longer time frame, during which some necessary data organisation and tool improvements could have been conducted.

The number of interviews and the sampling of the interviewees was quite small for a qualitative study, which makes the validity of this study slightly questionable. Preferably, the number of interviews should be bigger, and several representatives of each stakeholder should be interviewed for credibility. However, in a relatively small organisation such as M-Files, there is usually only one manager at each level. Therefore, the sampling of interviews could have only been extended to include more stakeholders, but not to contain more representative from the same profession on the managerial level. As the scope

of this study was limited to developing metrics targeted mainly for the test management, extending the interviews stakeholders selection to people whose job descriptions are further away from the test management would probably not have been feasible. Still, the number of interviewed testers and software developers could have been bigger. However, the other testers' input could be considered being taken into account through the informal discussions and questions during the study. On the other hand, the software developers' input is not as crucial as testing metrics do not play a big role in their job description.

Looking back at the metrics development process, it could have had more iterations. It was observed that the interviewees were often not able to phrase out more than an idea of what they wanted. However, only after developing the initial metrics, they had more specific requirements and development ideas, as well as some new ideas for metrics. This is understandable as it is very likely that the interviewees were not sure of what they wanted. In the reviews, when they had something concrete to grasp and they got a hint of what kind of metrics can be developed, it was easier for them to recognise what they wanted. While understanding this, the metrics identification could have been planned more iterative from the beginning.

The metrics identification and prioritisation were also highly influenced by the wishes of the test team manager and the senior director of R&D. In addition, the importance of concrete results was heavily weighted over conceptual development of metrics ideas. Therefore, the developed metrics and their format were mostly directed by those forces. It meant that during this study, there was not much room for deeper analysis over the metrics requirements, which is often highlighted as an important part of good metrics development practices. In a practical environment and with a tight schedule, this is to be expected.

6.3 Future development suggestions

During this study, several chances for improvement were identified. The nature of recognised development suggestions varies from little tweaks to test or defect management tools to high level development ideas for metrics. One of the development tasks is further developing the reports created during this study. It would be important to address the improvements suggestions and problems identified during the development of reports. The observations were described in detail in Section 5.4. While improving the current metrics would be good, it would be important to also develop better metrics. Metrics development goals can act as a good motivation and guidance for improving the organisation to support such metrics.

Overall, there were many improvement ideas for both test management and defect management tools. Examples include automating testing time estimation, improving the metadata mapping of test automation status of test cases, the test environment labelling in test runs, and the environment labelling of defects. Defect reports could be improved to include information of the stakeholder who found the issue, the product development phase the defect was discovered, and the testing method used to discover it.

From the interviews and discussions, it was noted that finding reliable ways to prioritise a feature in testing would be highly useful. This could include measuring factors such as complexity, size, novelty, amount of testing, the number and severity of found defects, the number and severity of customer tickets, and others. This links to the suggestion, that was presented in the interview of the senior quality assurance manager, of developing a risk model to which the test plan could be based on. A risk model could take into account the novelty, the developing party (subcontractor or internal), the amount of found defects so far, the effectiveness of testing, customer complaints, and others. Risk models and different risk analyses have become more common in the business, and this could be considered as a target in developing the quality assurance in the organisation. Developing a risk model into the organisation could be a more higher level target in the future development effort.

However, to be able to measure these things, at least a centrally managed list of features would be needed so that test cases, defects and customer tickets could be mapped to correct features consistently. Currently, these objects reside in different tools and their feature labelling differs from each other. Additionally, the feature list is lacking or contains duplicates even within one tool. This improvement would make the feature labelling more useful than just a hint of the context the object is about.

Having a higher level target and developing the processes, tools, and metrics step by step towards that goal would follow the recommendations made in the literature to start from simple and moving towards more advanced metrics (IEEE 1998, Chen et al. 2004, Kan 2002). The processes and used tools would probably be improved as a by-product of developing metrics, when the infrastructure is modified to support metrics.

7. CONCLUSIONS

The objective of this study was to improve the monitoring and reporting capabilities of testing in the case organisation with testing metrics. The literature review conducted in this study revealed that there are no ready-made solutions for metrics programs for organisations. Instead, the metrics should be developed based on the needs of the organisation and the amount of experience that the organisation has with metrics. The literature introduces various testing metrics developed for the software development industry.

In this thesis, the metrics development process partially applied the IEEE process standard for developing software quality metrics (IEEE 1998). Semi-structured interviews and periodical reviews were used to identify and analyse the metrics requirements, their purpose, and implementation details. M-Files Reporting feature was used to implement the identified metrics. As a result, a set of basic and most needed metrics were identified and implemented in reports. Therefore, the primary goal of this study was met. The developed reports are the current test run and pass rate, test run and pass rate trend, the number of test cases by state, the amount of testing with different web browsers and different operating systems in selected test plans, remaining testing time by test plan, and the number of executed test cases each week.

Not all identified metrics requirements could be implemented due to a limited time frame for the study, the lack of tool support, and the unavailability of needed data for the metric. However, the uncompleted requirements are a good source of improvement ideas. This study has also taught the organisation about the metrics development process which provides a good basis for future development. It could be concluded that also the secondary goal of this study was met with an additional benefit of learning about the metrics development process.

REFERENCES

Agarwal, B. B., Tayal, S. P. & Gupta, M. (2010). Software engineering and testing, Jones and Bartlett publishers, LLC. 516 pp.

Albrecht, A. J. & Gaffney, J. E. (1983). Software function, source lines of code, and development effort prediction: a software science validation, Software Engineering, IEEE Transactions on pp. 639–648.

Bach, J. (2003). Exploratory testing explained. Available: http://www.satisfice.com/articles/et-article.pdf. Accessed on 27.02.2015.

Bach, J. (2015). Session-Based Test Management. http://www.satisfice.com/sbtm/. Accessed on 27.02.2015.

Basili, V. R. & Weiss, D. M. (1984). A methodology for collecting valid software engineering data, Software Engineering, IEEE Transactions on **SE-10**(6): 728–738.

Black, R., McKay, J., Bath, G., Friedenberg, D., Homès, B., Onishi, K., Smith, M., Thompson, G. & Yumoto, T. (2012). ISTQB Certified tester advanced level syllabus - test manager. 82 p. Available: http://www.istqb.org/downloads/viewcategory/46.html. Accessed on 31.12.2014.

Chen, Y., Probert, R. L. & Robeson, K. (2004). Effective Test Metrics for Test Strategy Evolution, Proceedings of the 2004 Conference of the Centre for Advanced Studies on Collaborative Research, pp. 111–123.

CollabNet (2015). ScrumWorks Pro Web Site. http://www.collab.net/products/scrumworks. Accessed on 02.04.2015.

Craig, R. D. & Jaskiel, S. P. (2002). Systematic software testing, Artech House. 536 p.

Crispin, L. & Gregory, J. (2009). Agile testing: A practical guide for testers and agile teams, Pearson Education. 533 p.

Dutta, S., Lee, M. & Van Wassenhove, L. (1999). Software engineering in Europe: a study of best practices, IEEE software **16**(3): 82–90.

Gao, L. (2011). Research on implementation of software test management, 2011 3rd International Conference on Computer Research and Development, Vol. 3, pp. 234–237.

Garrett, T. (2011). Useful Automated Software Testing Metrics, Software Testing Geek .

IEEE (1998). IEEE Std. 1061-1998(R2009), Standard for a software quality metrics methodology. 20 p.

IEEE Computer Society (2008). IEEE 829-2008: Standard for Software and System Test Documentation. 118 p.

ISO (2008). ISO 9001: 2008. Quality management systems - Requirements. 27 p.

ISO (2011). ISO/IEC 25010:2011 Systems and software engineering–Systems and software Quality Requirements and Evaluation (SQuaRE)–System and software quality models. 34 p.

ISO/IEEE/IEC (2013). ISO/IEC/IEEE 29119-1: Software systems engineering - software testing - part 1. 56 p.

Johnson, H. T. (2006). Lean Dilemma: Choose System Principles or Management Accounting Controls, Not Both. Available: http://in2in.org/insights/TomJohnson-LeanDilemma.pdf. Accessed on 24.01.2015.

Jones, C. (1994). Software metrics: good, bad and missing, Computer **27**(9): 98–100.

Jones, C. (2008). Applied Software Measurement: Global Analysis of Productivity and Quality, 3rd edn, The McGraw-Hill.

Jones, C. (2012). Project Performance International Resources. Available: http://www.ppi-int.com/systems-engineering/free%20resources/Software%20Quality%20Metrics%20Capers%20Jones%20120607.pdf. Accessed on 24.01.2015.

Jones, C. (2014). The Mess of Software Metrics. v. 2.0, Available: http://namcookanalytics.com/wp-content/uploads/2014/09/problems-variations-software-metrics.pdf. Accessed on 09.04.2015.

Kan, S. H. (2002). Metrics and models in software quality engineering, 2nd edn, Addison-Wesley Longman Publishing Co., Inc.

Kaner, C. & Bond, W. P. (2004). Software Engineering Metrics: What Do They Measure and How Do We Know?, In METRICS 2004. IEEE CS, Press.

Kivimäki, J. (2007). Testausprojektin mittarit. 43 p., Unpublished presentation.

Lazic, L. & Mastorakis, N. (2008). Cost effective software test metrics, WSEAS Transactions on Computers **7**(6): 599–619.

Lincke, R., Lundberg, J. & Löwe, W. (2008). Comparing software metrics tools, Proceedings of the 2008 international symposium on Software testing and analysis, ACM, pp. 131–142.

Lodhi, A., Wind, S. & Turowski, K. (2013). Test Management Framework for Managing IT Projects in Industry, e-Business Engineering (ICEBE), 2013 IEEE 10th International Conference on, IEEE, pp. 509–514.

M-Files Corporation (2013). Product Development Process description #1094 ENG v.1.3 Main Process.

M-Files Corporation (2014a). M-Files Reporting - Getting Started Guide (Internal). Version 55. Accessed on 07.03.2015.

M-Files Corporation (2014b). Product Development SOP #1143 ENG v.1.0 Tracker Instructions.

M-Files Corporation (2014c). Product Development SOP #1204 ENG v.1.0 Test Strategy of M-Files.

M-Files Corporation (2015a). M-Files Web Site. http://www.m-files.com/en. Accessed on 19.02.2015.

M-Files Corporation (2015b). Test Management tool on M-Files. Version 21, Accessed on 20.02.2015.

Microsoft Developer Network (2008a). Introducing Business Intelligence Management Studio. https://msdn.microsoft.com/en-us/library/ms173767(v=sql.105).aspx. Accessed on 09.04.2015.

Microsoft Developer Network (2008b). Report Builder (SSRS). https://msdn.microsoft.com/en-us/library/hh213578.aspx. Accessed on 09.04.2015.

Microsoft Developer Network (2008c). SQL Server Reporting Services. https://msdn.microsoft.com/en-us/library/ms159106(v=sql.105).aspx. Accessed on 09.04.2015.

Mills, E. E. & Shingler, K. H. (1988). Software Metrics – SEI Curriculum Module SEI-CM-12-1.1.

Müller, T. & Friedenberg, D. (2011). ISTQB Certified tester foundation level syllabus. 78 p. Available: http://www.istqb.org/downloads/viewcategory/16.html. Accessed on 26.01.2015.

NUnit.org (2015). NUnit Web Site. http://www.nunit.org/. Accessed on 09.04.2015.

Parveen, T., Tilley, S. & Gonzalez, G. (2007). A Case Study in Test Management, Proceedings of the 45th Annual Southeast Regional Conference, ACM-SE 45, ACM, New York, NY, USA, pp. 82–87.

Pohjoisvirta, L. (2013). Choosing a tool for improved software test management, Master's thesis, Tampere University of Technology.

Saunders, M., Lewis, P. & Thornhill, A. (2009). Research methods for business students, 5 edn, Pearson Education Limited.

TMMi Foundation (2015). TMMi Web Site. http://www.tmmi.org/. Accessed on 26.02.2015.

Vuori, M. (2014). About metrics and reporting in model-based robot assisted functional testing. Project report. 13 p.

Walia, M. (2012). Realizing Effficiency & Effectiveness in Software Testing through a Comprehensive Metrics Model. Infosys white paper. Available: http://www.infosys.com/engineering-services/white-papers/Documents/comprehensive-metrics-model.pdf. Accessed on 14.01.2015.

APPENDIX A: THE LIST OF INTERVIEWEES

Reference	Interviewee
I1	Senior Director of Research and Development
I2	Test Manager
I3	Senior Software Designer / a former Test Manager and Tester
I4	Senior Quality Assurance Manager

APPENDIX B: METRICS REQUIREMENTS IDENTIFIED IN THE INTERVIEWS

ID	Metric requirement	Purpose	Interview
R1	I want to see the pass rate of system testing now and over time	I want to see the results of testing.	I1
R2	I want to compare the system testing pass rate of different releases	I want to compare the results of testing from release to release.	I1
R3	I want to see the run rate of system testing now and over time by release	I want to see the progress of testing and how much there is left	I1
R4	I want to know how many test cases are in Draft state and need to be reviewed in the test management vault.	I want to approximate how much effort would be needed to review the new test cases created by the subcontractor.	I2
R5	I want to see the number of test cases currently, and the difference in a specific time frame	I sometimes need to report the number of test cases now and how much it has changed from release to release.	I2
R6	I want to see more easily what is the distribution of used/planned environments, namely web browser and operating system, in testing.	I want to better ways to see if various environments (web browser and operating systems) are used in testing in the desired ratio, e.g. as their relative usage in the world.	I2
R7	I want to see when a certain test phase is done with the current estimates	I want to know then the testing is done.	I2
R8	I want to see how many test cases are executed by the testing team weekly.	I want to know the testing pace in calendar time.	I3
R9	I want to see the number of test cases classified to e.g. automated, automatable, and manual	I want to estimate the required manual testing effort	I2

ID	Metric requirement	Purpose	Interview
R10	I want to see when a test set has been last run	I want to make sure that all test sets are run at least sometimes if not in every system testing phases.	I2
R11	I want a view that allows to browse the test runs under a specific test plan and see their states.		I1
R12	I want test scheduling and monitoring to be easier		I2
R13	I want to know which features had a lot of defects during user story testing.	I want to know which features should be prioritized during the system testing	I2
R14	I want to try pairwise testing metrics to see if some features break in pairs		I3
R15	I want a risk model to be developed first, the testing to be based on it, and the metrics to measure the testing effort in relation to the risk model, in which e.g. the risks are measured and testing is prioritised based on the total risk.	I want the metrics to be based on a risk model, that would help meet the customers expectations from the organisation as their software vendor	I4